Craziesky24: Artwork Created by Drones

By: Amber Miller*, Atlas Karm*, Adil Gill*, Rayan Syed*
University of Illinois Urbana Champaign, Grainger College of Engineering

I. Abstract

To create more STEM engagement, it is beneficial to create connections with other fields, such as art. Utilizing a stable controller and observer, a drone with an LED ring deck payload, creates 2D art work based on its flight path through an echo effect. To create the flight path, Bitmap takes a sketch in Python and turns it into a path that an algorithm optimizes the drone to fly through. While the initial goal o fusing some photosensitive paper or glow in the dark medium was to be used, video effects that use an echo effect show the drone's full trajectory. An optimized controller and observer were designed so that the quadcopter could fly through the tracked path. Art is expressed in many forms, and the intersection of art and STEM brings more engagement in both fields. This project topic may inspire students in the local CU community to learn about how STEM can be utilized for art. Adjustments from the standard parameters within the equations of motion are implemented as the addition of an LED ring deck increases mass and changes the moments of inertia. Discretization and curve fitting will be applied to the desired path. Utilizing the Talbot Drone lab, the motion capture system of the room is utilized for the drone to observe its position and orientation. The connections people will make between stem and art through this project will inspire both areas of study and enjoyment.

Nomenclature

x	=	State
u	=	Input
v_x, v_y, v_z	=	Velocity components in the x, y, and z directions
w_x, w_y, w_z	=	Angular velocity components about the x, y, and z axes
a_z	=	Acceleration in the z direction
ϕ	=	Roll angle (rotation about the x-axis)
θ	=	Pitch angle (rotation about the y-axis)
ψ	=	Yaw angle (rotation about the z-axis)
g	=	Acceleration due to gravity
cos, sin, tan	=	Trigonometric functions cosine, sine, and tangent
$RMSE$	=	Root Mean Square Error
τ_x, τ_y, τ_z	=	Torques applied about the x, y, and z axes to control orientation
f_z	=	Thrust force applied along the z-axis to control altitude
m	=	Mass of the drone
J_x, J_y, J_z	=	Moments of inertia about the x, y, and z axes
g	=	Gravitational acceleration
k_{flow}	=	Flow constant
$\dot{x}$	=	Time derivative of the state vector, representing the rate of change of the state
A	=	State matrix, representing the linearized dynamics of the system
B	=	Input matrix, representing how the input affects the system's dynamics

II. Introduction

Utilizing Crazyflie hardware and software a stable controller and observer is used to control the drone through a predrawn trajectory. This type of flight tracking is akin to aircraft sky art posted on Flightradar24. Art is expressed in

*Undergraduate Student, University of Illinois Urbana-Champaign, Urbana, Illinois, 61801 many forms, and the intersection of art and STEM brings more engagement in both fields.

The field of quadcopter dynamics is a widely studied field. The use of four motors allows for a direct connection between the three torques and one net force in the z body axis. The torques are modeled with 3 equations of motion with one equation of motion for the net force relates to the four individual motor commands which successfully guarantees a full rank system. Such systems are able to be analytically solved. These systems in control theory are often expressed in the state space model, and this model allows for the use of approximating the change in state as purely a function of the current state, an input, and the observer output. A paper that may be utilized for future reference in how controller designs are implemented is shown through the following citation [1]. This paper also goes into a slightly different way to express controller design than what is utilized in University of Illinois coursework. A paper that goes into more detail than the previous in regards to motor commands and system design is shown in the paper, "Trajectory Tracking of a Quadcopter UAV with Optimal Translational Control" [2]. Here they show the full relationship between dynamics and motor commands, and in this paper they show how this system can be utilized to control a drone through a set of parametrized curve in 3D space. While this project may not go through 3D space, the curves utilized for randomness of a user allow for more further complication and investigation about other avenues to model a drones desired trajectories.

Utilizing similar methods a paper titled "Trajectory Optimization with Optimization-Based Dynamics" [3]. This article provides the information to equip a quadcopter drone with trajectory tracking strategies. The article demonstrates that in order to minimize the energy consumption by the rotors, a linear optimal control method must be implemented. To connect this project form ones previous, the Fall 2022 paper titled, "Trajectory Computation and Optimization of a Crazyflie 2.0" will plan to be referenced [4]. Their paper went over optimization for applying limitations and speed requirements based on given trajectories. For avoidance of obstacles there are certain paths that have specific impacts on the acceleration for parametric curves. A sharp turn typically characterizes a large acceleration, and in this project being able to constrain its acceleration so that it does not go beyond limits that make this drone less controllable.

This project topic may inspire students in the local CU community to learn about how STEM can be utilized for art. The drone will carry an additional payload of an LED deck. Utilizing pygame to create a drawing interface, a sketched trajectory gets translated into a discretized map of x and y coordinates. Curve fitting is applied to smooth out the edges and to create a parametric function that gets transformed into a CSV file that the move smooth function uses. To track the x and y coordinates on the drone, the active mo cap system in the Talbot drone lab will be utilized. It is challenging to track the x and y coordinates in a place without an active marker, so some assurance of position is required to accurately create the artwork. An example of such sky art comes from an article posted on Flightradar24 [5].

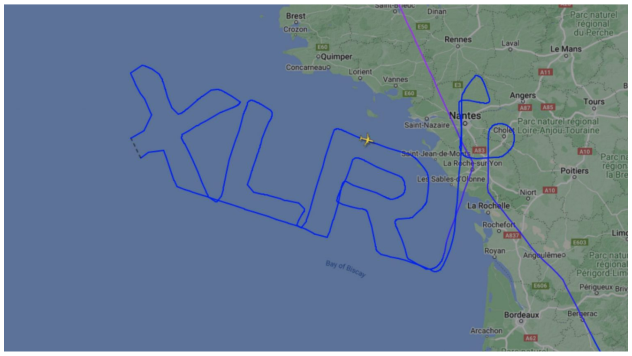


Fig. 1 Sky art on Flightradar24

Our idea for this project was influenced by sky art on Flightradar24 which is an online real-time flight tracking software [5]. Pilots usually perform these flights for fun, art or publicity to trace out designs, logos, animals or objects. Sky art requires careful planning and execution as pilots need to follow specific way points to draw the design in the sky.

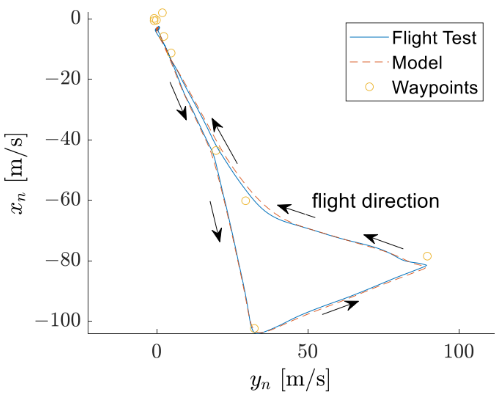


Fig. 2 A 2D trajectory on the pygame interface

Moreover, trajectory tracking is important for Unmanned Aerial Vehicles (UAV) as well and this enables them to follow a pre-defined path in space as shown in the graph above [6]. This is applicable in many other areas as well apart from art such as logistics, agriculture, and environmental monitoring. In our project the Crazyflie quad copter will fly a sketched path with accuracy while carrying a LED deck that visibly traces the flight path. The initial trajectory will be generated using a pygame interface which allows users to sketch a path on the screen. The interface stores the x and y coordinates of the path as a series of way points. Since these hand drawn paths can be challenging for the drone to follow Bezier curves can be used to convert them into smooth, continuous paths. The new smooth trajectory follows the general shape drawn by the user but it eliminates sharp turns. This makes the quad copter's flight more stable and reduces the strain on the motors. This project demonstrates the the unconventional uses of a drone and how technical concepts in autonomous systems can connect with artistic expression. This can encourage students in the Champaign Urbana community to consider how STEM fields can affect art and culture.

III. Theory

A. Motor Commands

To predict the behavior of the drone, it was important to find the forces and the moments acting on it. Hence, to determine these, the group used the parameters of the motor command to find the forces and moments:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \\ f_z \end{bmatrix} = \begin{bmatrix} -lk_F & -lk_F & lk_F & lk_F \\ -lk_F & lk_F & lk_F & -lk_F \\ -k_M & k_M & -k_M & k_M \\ k_F & k_F & k_F & k_F \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} \tag{1}$$

The first matrix represents the torques in the x,y,z directions, as well as the force in the z direction, while the second matrix is the constant matrix, wherein k_F is the coefficient of the force, k_M is the coefficient of the moment, while l is the length from the center of mass of the drone to the point where the moment is acting.

B. Controller Design

The dynamic model for the drone was created using the linearized state space model. The equations of motion have the form

$$\dot{s} = f(s, i, p)$$

where the state vector s contains variables that define the current condition of the drone at any given moment, the input vector i contains variables representing external forces, and the parameter vector p includes constants specific to the drone's physical properties.

$$s = \begin{bmatrix} p_x \\ p_y \\ p_z \\ \psi \\ \theta \\ \phi \\ v_x \\ v_y \\ v_z \end{bmatrix}, \quad i = \begin{bmatrix} w_x \\ w_y \\ w_z \\ a_z \end{bmatrix}, \quad p = \begin{bmatrix} g \\ k_{\text{flow}} \end{bmatrix}. \tag{2}$$

The equations of motion can be defined as:

$$\dot{s} = \begin{bmatrix} v_x \cos(\psi)\cos(\theta) + v_y\left(\sin(\phi)\sin(\theta)\cos(\psi) - \sin(\psi)\cos(\phi)\right) + v_z\left(\sin(\phi)\sin(\psi) + \sin(\theta)\cos(\phi)\cos(\psi)\right) \\ v_x \sin(\psi)\cos(\theta) + v_y\left(\sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi)\right) + v_z\left(-\sin(\phi)\cos(\psi) + \sin(\psi)\sin(\theta)\cos(\phi)\right) \\ -v_x\sin(\theta) + v_y\sin(\phi)\cos(\theta) + v_z\cos(\phi)\cos(\theta) \\ \frac{w_y\sin(\phi)}{\cos(\theta)} + \frac{w_z\cos(\phi)}{\cos(\theta)} \\ w_y\cos(\phi) - w_z\sin(\phi) \\ w_x + w_y\sin(\phi)\tan(\theta) + w_z\cos(\phi)\tan(\theta) \\ g\sin(\theta) - v_y w_z + v_z w_y \\ -g\sin(\phi)\cos(\theta) + v_x w_z - v_z w_x \\ a_z - g\cos(\phi)\cos(\theta) \end{bmatrix} \tag{3}$$

The next step in deriving the state space model was to set the equilibrium points. The following equilibrium points were chosen:

$$s_{eq} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ z_{eq} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad i_{eq} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ g \end{bmatrix}, \quad p = \begin{bmatrix} g \\ k_{\text{flow}} \end{bmatrix}. \tag{4}$$

The steady-state equation $\dot{x} = Ax + Bu$ is used to define the equations of motion for the system. In this equation, x represents the state vector, which describes the system's current condition, and u represents the input vector, which includes the control inputs applied to the system. The matrices A and B are the state and input matrices, respectively, which define how the state evolves based on its current value and the applied inputs. This equation forms the foundation for analyzing and designing controllers for dynamic systems.

$$A = \left.\frac{\partial f}{\partial s}\right|_{s_{eq}, i_{eq}, p_{eq}}, \quad B = \left.\frac{\partial f}{\partial u}\right|_{s_{eq}, i_{eq}, p_{eq}} \tag{4}$$

The following matrices were obtained:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g \\ 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The next step in designing an effective model for the drone was to design a better controller using the Linear-Quadratic Regulator (LQR) model to find the K matrix. The LQR is an optimal control method used to determine the control input $u(t)$ that minimizes a given quadratic cost function while ensuring the system dynamics are satisfied. It is widely applied in control systems for achieving optimal performance with minimal energy usage. The goal of the LQR method is to find the feedback gain matrix K such that the control input

$$u(t) = -Kx(t) \tag{6}$$

is the optimal solution to the following optimization problem:

$$\begin{aligned} \text{minimize}_{u(t)} \quad & \int_{t_0}^{\infty} \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) dt, \\ \text{subject to} \quad & \dot{x}(t) = Ax(t) + Bu(t), \\ & x(t_0) = x_0. \end{aligned} \tag{7}$$

C. Observer Design

The drone required a new observer to complete the mission goals. Since the LED deck replaced the flow deck, this removed the observability of the velocity and z position measurements. To continue to attain a stable observer, the motion capture system in the drone lab was used to observe all positions and orientations. For the observer equations of motion, they are the same as the equations used in Eq. 5. The one difference is that the equations are centered around a new equilibrium point. This was significant for observer design with the flow deck because at $z = 0$ there is a singularity, so the point of equilibrium would change based on the equilibrium point. However, this does not make a difference for when using the motion capture system since it is able to directly observe the position and orientation. The observer directly measures the x_{mocap}, y_{mocap}, z_{mocap}, ψ_{mocap}, θ_{mocap}, & ϕ_{mocap}. The following equations show how the observer model views the states.

$$o = \begin{bmatrix} p_x \\ p_x \\ p_z \\ \psi \\ \theta \\ \phi \end{bmatrix}_{\text{mocap}} \qquad s = \begin{bmatrix} o_x \\ o_y \\ o_z \\ \psi \\ \theta \\ \phi \\ v_x \\ v_y \\ v_z \end{bmatrix} \qquad i = \begin{bmatrix} w_x \\ w_y \\ w_z \\ a_z \end{bmatrix} \qquad p = \begin{bmatrix} g \\ k_{\text{flow}} = 0 \end{bmatrix} \tag{8}$$

The next part of the drone that requires estimation is the velocities, $[v_x, v_y, v_z]$. These attributes are estimated through finite difference.

$$v_{i+1} = v_i + \frac{p_i - p_{i-1}}{dt} \tag{9}$$

To implement the observer design, the state is estimated with a closed loop L matrix: $\dot{x} = (A - BK)x - L(Cx - y)$. The L matrix is a 9×9 matrix that is found through the linear quadric regulator method. This is then rewritten for each observable component below.

$$p_x = p_x + dt\left[v_x - \left(L_{11}p_{x_{err}} + L_{15}\theta_{err}\right)\right] \tag{10}$$
$$p_y = p_y + dt\left[v_y - \left(L_{22}p_{y_{err}} + L_{26}\phi_{err}\right)\right] \tag{11}$$
$$p_z = p_z + dt\left[v_z - L_{33}p_{z_{err}}\right] \tag{12}$$
$$\psi = \psi + dt\left[w_z - L_{44}\psi_{err}\right] \tag{13}$$
$$\theta = \theta + dt\left[w_y - \left(L_{51}p_{x_{err}} + L_{55}\theta_{err}\right)\right] \tag{14}$$
$$\phi = \phi + dt\left[w_x - \left(L_{62}p_{y_{err}} + L_{66}\phi_{err}\right)\right] \tag{15}$$
$$v_x = v_x + dt\left[g\theta - \left(L_{71}p_{x_{err}} + L_{75}\theta_{err}\right)\right] \tag{16}$$
$$v_y = v_y + dt\left[-g\phi - \left(L_{82}p_{y_{err}} + L_{86}\phi_{err}\right)\right] \tag{17}$$
$$v_z = v_z + dt\left[a_z - g - L_{93}p_{z_{err}}\right] \tag{18}$$

Where each error term can be expressed as $s_{i_{err}}$, the errors are evaluated as the difference between the estimated state and the observed state: $s_{i_{err}} = s_i - s_{\text{mocap}}$.

D. Trajectory Design

When examining methods by which one could generate a dynamically feasible trajectory which approximates a user generated trajectory as closely as possible, a few considerations became apparent. For one, it became clear that a constraint on the velocity, and by extension acceleration and jerk, was necessary when designing a feasible trajectory, as without such a constraint, the drone would be unable to perform the high jerk maneuvers which may be required by an unconstrained trajectory. Therefore, a method would require some way to limit or control the overall curvature of the drone's path. One method explored which achieves this was the use of quadratic Bezier curves to approximate the user-inputted trajectory. Using bezier curves for the trajectory design was ideal because they are computationally inexpensive to produce, as well as to analyze. Because bezier curves are parametric, one can easily interpolate any value along the curve by using time as the parametric variable. In addition the derivatives can also be easily computed, alongside the path length and overall curvature. Additionally, by splitting the bezier into piecewise approximations with boundary constraints, further resources will be saved upon computation. A bezier curve can be defined as a mapping from $s \in [0, 1]$ to convex combinations of points $v_0, v_1, \ldots, v_n$ in some vector space which can be shown as [1]:

$$B(s) = \sum_{j=0}^{n} \binom{n}{j} s^j (1-s)^{n-j} \cdot v_j. \tag{19}$$

IV. Methods

A. Bitmap

This section examines the method by which an image drawn by a user was converted into a flight trajectory. In order to create the user interface which would allow the user to draw, a pygame interface was used. The initialization of the interface is seen in Fig. 7. The interface, allows users to draw a 1 pixel wide line upon holding a mouse-click. As shown in Fig. 8, the pygame interface records both the points drawn as well as the time at which they were drawn. In this way, the points can be ordered by the time they were drawn. This lends itself well to the creation of parametric curves to represent theses points. After the pygame interface is closed, the code in Fig. 9 will convert the pixels on the screen into a bitmap array which documents all the coordinates of the user drawn points in order of when they were drawn. With these coordinates saved, they can be re-used as nodes for the creation of a set of bezier curves. The code in Fig. 10 creates a while loop which goes through all the coordinates saved from the bitmap, and from every set of 10 points saved, maps a bezier curve onto the points where the 10 points now serve as nodes for the curve. Additionally, a dummy parametric variable which represents time will allow these bezier curves, which are currently objects in the code, to be interpolated at evenly timed points across the segment. In steps of 10 points at a time, the last point of the current segment is re-used as the first point in the upcoming segment because the bezier curve constrains itself such that the first and last node are always reached. This allows for the creation of a continuous segment of bezier curves representing the user drawn image. The interpolated points along the segments of these curves are saved in a new numpy array, which is then exported to an excel data sheet by using the code shown in Fig. 11.

B. Move Smooth Along Curve

This section focuses on the implementation of the trajectory developed by the bezier curves. In order to allow the drone the move through these interpolated coordinates of the bezier curve as smoothly as possible, the move smoothly command which had been written earlier in the semester was re-introduced, albeit with some minor changes. Firstly, because the drone utilizes the mocap coordinate frame, the move smoothly command needs to incorporate the initial position of the drone in the mocap coordinate frame. As a result, the new initialization can be seen in Fig. 12.

The new start point then needs to be passed through to the send position set point command. Additionally, it was noted that without the added sleep command to the else statement, due to the nature of the how closely defined the bezier curve interpolated points were to each other, the drone would skip through a series of segments because the drone could not distinguish between the small differences in horizontal positioning. By adding this sleep command to the else statement, the drone gives itself time to catch up to where it thinks it is in space. Manipulating the length of these sleep commands was the primary method by which the flight path was smoothed, in addition to allowing for more direct control of the velocity of the drone. These times were picked in such a way that the drone can continue through segments without any noticeable stopping period. These changes can be seen in Fig. 13

C. LQR

1. Optimal Controller

Linear Quadratic Regulator (LQR) is a minimization problem that seeks to minimize the total cost of a function based off the weightage given to effort, given by the inputs into the system, and error, given by the difference between the states of the system and the desired equilibrium point. Here is the definition for the LQR cost function for the controller:

$$\begin{aligned} \text{minimize}\, u_{[t_0,\infty]}, \quad & \int_{t_0}^{\infty} (x(t)^T Q x(t) + u(t)^T R u(t))dt \\ \text{subject to} \quad & \dot{x}(t) = Ax(t) + Bu(t), \quad x(0) = x_0 \end{aligned} \tag{20}$$

The input that achieves minimum cost is given by:

$$u(t) = -Kx(t) \tag{21}$$

where P is a matrix, found with

$$PBR^{-1}B^T P - PA - A^T P - Q = 0 \tag{22}$$

and where K, the gain matrix, is found by solving the bottom equation.

$$K = R^{-1}B^T P. \tag{23}$$

The values of Q and R are chosen using Bryson's rule. The diagonal elements of Q and R are set to be the inverse square of the desired max value for each component of state and input. The diagonal elements of Q correspond to the weights for each state while the diagonal elements of R correspond to the weights for the input on each element. Where d is the diagonal element and $q_{max} = [m_{max} \quad n_{max}]$ is the desired maximum value, the elements along the diagonal follow this relationship: $d = (q_{max})^{-2}$.

2. Optimal Observer

For an observer to be optimized, a similar method as above where is implemented ti attempt to minimize an expression subject to a condition. The method used to find the most optimal observer uses the Kalman Filter problem.

The function to minimize is below.

$$
\begin{aligned}
\text{minimize}\, n_{[t_0,\infty]}, d_{[t_0,\infty]}, \quad & \int_{t_0}^{\infty} (n(t)^T Q_o n(t) + d(t)^T R_o d(t)) dt \\
\text{subject to} \quad & \dot{x}(t) = Ax(t) + Bu(t) + d(t), \quad x(0) = x_0 \\
& y(t) = Cx(t) + n(t)
\end{aligned}
\tag{24}
$$

Here the value $n(t)$ represents the noise of the measurements and $d(t)$ represents the disturbances, so Q_o attempts to minimize the noise of the observer, while R_o attempts to minimize the disturbances of the measurements. This leads to a solution for the L. The initial guesses for Q_0 and R_0 follow Bryson's rule, such as the case with the controller design. The one difference here is that the parameters that undergo squared rationalization is the standard deviation in the measurements.

p_x	p_y	p_z	ψ	θ	ϕ	v_x	v_y	v_z
0.028	0.033	0.004	0.024	0.040	0.025	0.035	0.027	0.052

Table 1 Standard Deviations

Using Bryson's rule: $Q_{ii} = \frac{1}{\sigma_i^2}$, produced a well observable drone. Though, edits were made the R_0 matrix so that errors could be minimized. Due to the natural imperfections in the drone, fine tuning was required to change how it would react to errors in the state.

V. Result

To preface the results, there were three main requirements that the crazyflie drone had to meet. With the new equations of motion, the drone shall be able to fly a square path with the LED deck below the drone. Once the bitmap code was complete, the drone shall be able to take in those points and use it to move the drone smoothly through the predrawn trajectory. Finally, to have certainty about the integrity of the image exposed during the flight, the drone shall meet standard deviations of 0.5 in the x and y components, and it shall not exceed a standard deviation of 5 degrees in yaw.

A. Flight Path

As seen in Fig. 3, the flight path for the drone, as captured by the mocap system, overlaps very well with the desired flight path. The tuning of both the controller and observer were crucial in order for this to occur, and in addition, this demonstrates the robustness of the move along curve command outlined earlier. Clear lettering can be distinguished. This was demonstrated most visibly by the results of the group's drone video presentation, in which an echo effect was used to overlap frames from a video of the drone flying in a similar, but not exact, flight path. A frame from the end of the video as shown in Fig. 4 demonstrates the flight path of the drone visually.

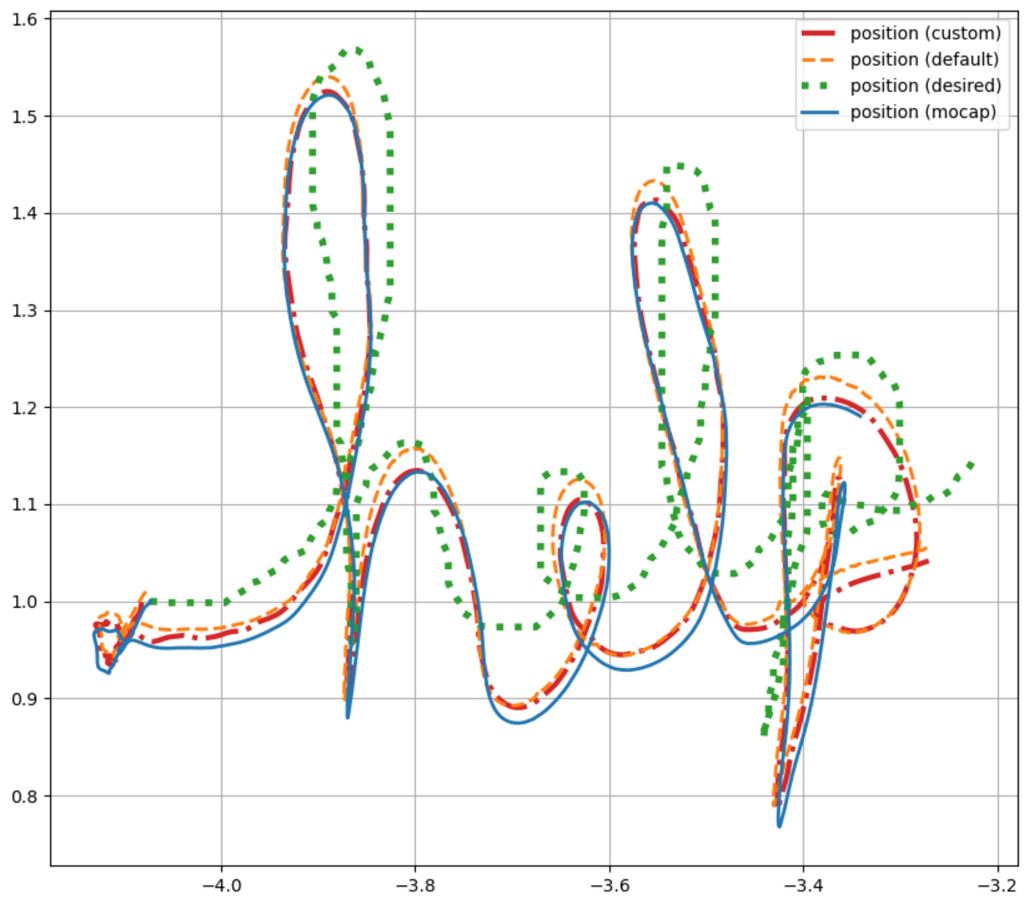


Fig. 3 Trajectory Compared to Desired Trajectory

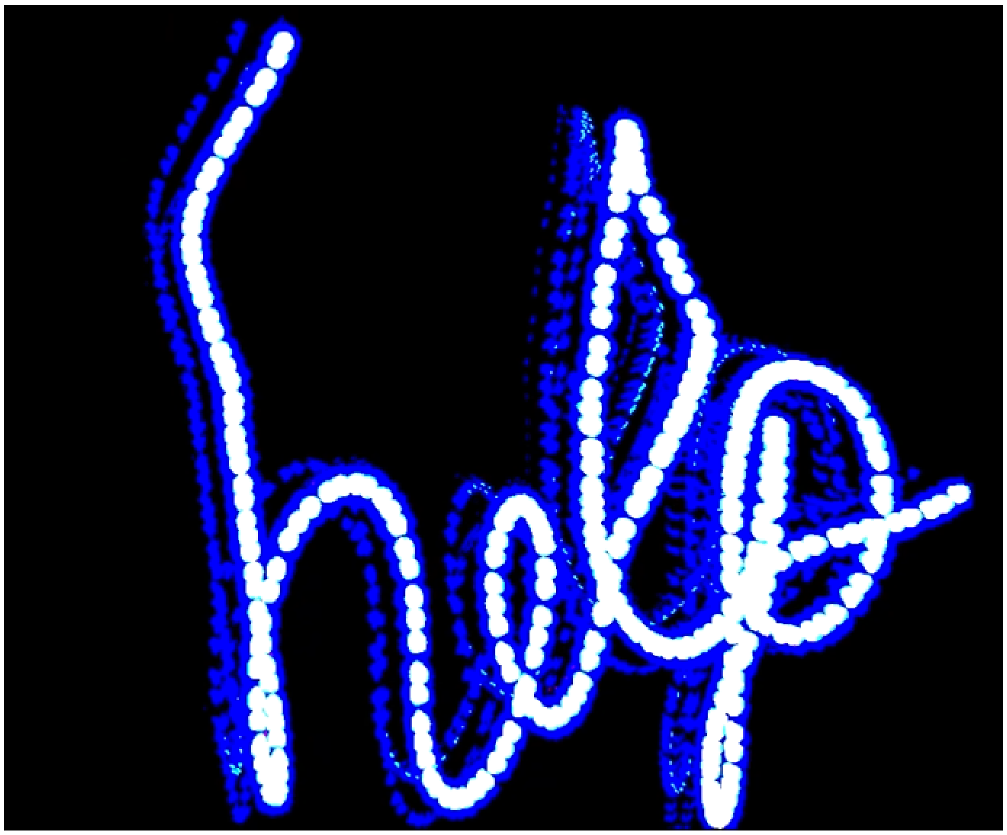


Fig. 4   Path of Drone Demonstrated With Echo Effect

B. Standard Deviation of Error

Upon analysis of the flight path it can be seen that the mocap position and desired position are close, but slightly offset. Since an offset would not disfigure the image the drone is trying to draw, it becomes necessary to analyze the error in desired versus actual positioning by looking at the standard deviation of error between the mocap data and the desired position data. For the flight path given above, the errors for our x,y, and z states of the drone are shown in Fig. 6 alongside the standard RMSE and mean error, shown in Fig. 5.

Parameter	RMSE	Mean	Standard Deviation
p_x	0.115	-0.072	0.090
p_y	0.243	-0.070	0.233
p_z	0.187	0.184	0.037
psi	2.597	1.305	2.245

Fig. 5 Error in Standard Deviation

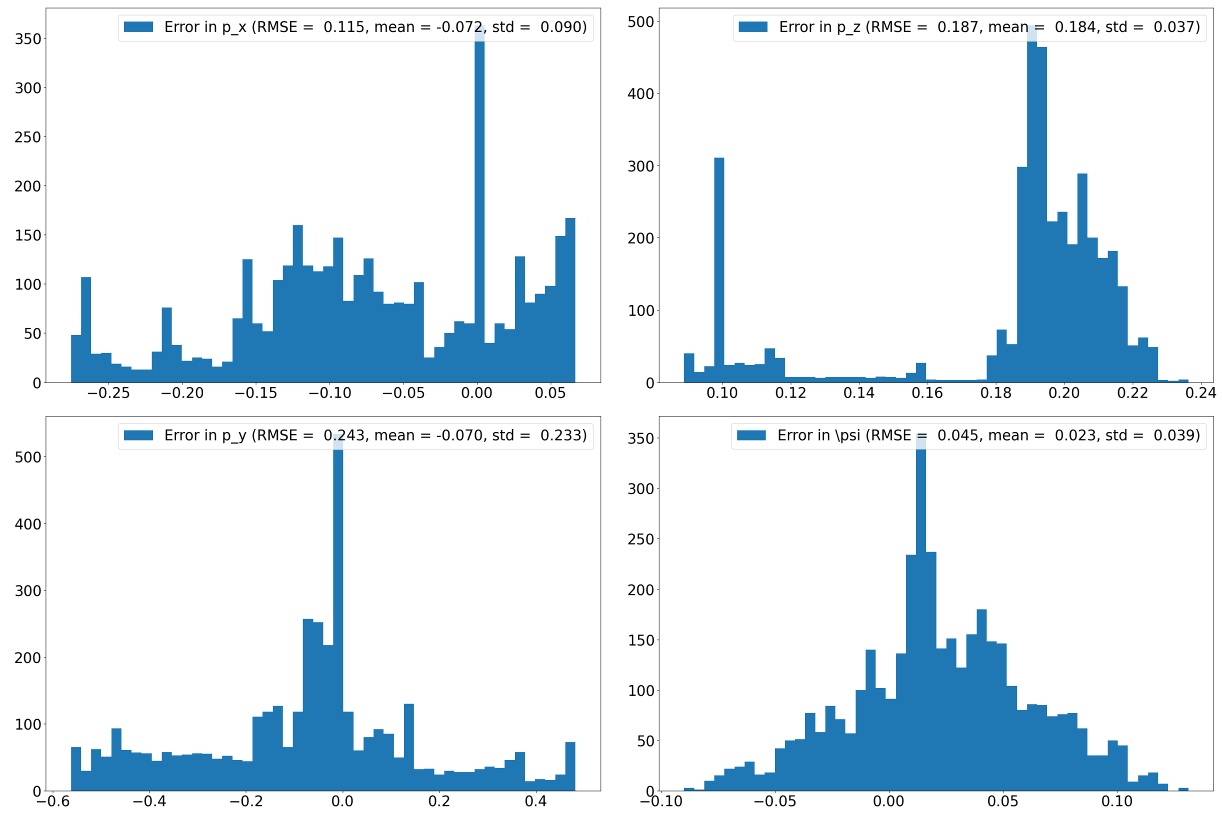


Fig. 6 Error in States

VI. Discussion

A. Challenges Faced

During the flight test, the group had encountered multiple challenges to implement the desired flight outcome and accomplish all the goals of the milestones. One of the main challenges was the battery power, and it made the group reconsider the flight path of the drone. Initially, the group had planned the star shaped object as the drones' trajectory, but due to battery charge limitations, which had the maximum charging capacity of up to 4 volts, the group had to change its decisions for a proper flight pathway that minimized the battery's power consumption while also making a successful trajectory for the drone while also ensuring that it performed a task that might be helpful for the local community, and hence it was decided to make the drone draw the word "help" in cursive letters as its word command. Another great challenge that was faced during the implementation of our flight plan was that the drone was unable to move due to a problem with the custom controller. Initially this was considered to be an issue with the motor or with the propeller. However, after a few more tests, it was found that all of the drone's hardware was working properly and the problem mainly was with the custom controller. After the diagnosing the issue, the group revisited previous lab sections and modified the code for the custom controller in order to make the drone working. Another challenge that was noticed during our flight implementation was that the group encountered errors working in the drone lab when other people's drones were flying. Due to numerous drones being flown at the same time, our drone had to fly beyond the premises of the testing carpet, and this gave rise to errors in the measurements, and the only way forward to fix this issue was to fly the drone around the middle of the carpet and make the trajectory such that it did not create any errors in the measurement.

B. Troubleshooting

The main trouble throughout the course of the project is working within real world systems and in fine tuned systems. In controller and observer design, fine-tuned parameters are required or otherwise the system may be unstable. Theoretically, any values for Q and R should produce a stable controller and observer, but that is only valid for the $t \to \infty$ case. The system of a quadrotor is more complicated, and this means that strong perturbations have lead to uncontrollable orientations.

The Crazyflie drone is not a perfect object. Ultimately, the drone is sensitive to its surrounding, attachments, and radio connections. Often times in the event of a collision, the motors, bus, battery, or sensors could get damaged. During the project, following a preflight procedure is needed as to ensure that the quadrotor would be able to take-off. Many tricks that were employed involved power cycling, recharging the drone, putting the drone onto the charger (even if it was full), replacing motors that sounded off or failed motor tests, and re-flashing the firmware. All of these parts helped with ensuring the drone could start up with minimal potential for hardware issues.

The final part of conflict resolution was communication with others in the drone lab. As people would enter the lab, there was uncertainty with regards to their motion capture and radio numbers. If two drones were using the same motion capture number, then one drone may record the data from the other drone. This required the need for improved communication between all students who were sharing the lab. While working within a group proves its own challenges, there is much growth to be had with people who are working outside a designated group.

C. Connection to Goal

Our project serves as an example of how observer theory, control theory, and drones can all be used to produce creative expression.The motion capture technique at the Talbot Drone Lab guarantees accuracy, enabling the drone's movement to smoothly follow the intended artistic trajectory. This junction involves the local CU community by showcasing how engineering concepts can support and enrich creative endeavors, while also highlighting the value of STEM in non-traditional applications. With this initiative, there is hope to stimulate creativity and curiosity in students and community members, inspiring them to further explore the imaginative possibilities of technology.This program is especially well-suited for aerospace outreach since it offers students and the local CU community an easy and interesting way to engage with technical ideas in a practical and eye-catching way. Stakeholders want to increase the interest in aerospace engineering and highlight how it may spur innovation in unanticipated fields through hands-on demonstrations and the production of aesthetically striking art forms.

VII. Conclusion

Throughout this project the drone satisfied the initial requirements to be controlled using the LED deck, take in Bezier curves for path tracking, and it accomplished these goals with standard deviations within the specified requirements. The goal of the project was to make the drone successfully fly within a pre-drawn trajectory with an crazy file LED deck, wherein only one of the twelve lights were lit while the drone flew over a cardboard that was painted with spray paint that would glow in the dark. However, that initial artistic goal was unmet due to not enough time for the exposure to have an impact on the canvas. The main constraint during the implementation of this flight was the issue with the battery, as the the group had to reconcile between conveying a beneficial message that had a positive impact on the community through the drone while also ensuring that the drone was sufficiently charged throughout the flight, a lack thereof resulted in the drone following offset trajectories. Another important consideration made during the flight test was that constraints had to be put on velocity and its higher order derivatives, such as the acceleration and the jerk. For this reason, the trajectory of the bezier curves, a category of parametric curves, was utilized. Due to their computationally inexpensive nature, making a trajectory design was significantly more feasible. In order to make the drone to fly at a given trajectory, the move smoothly command was used. Through this project, the group learned how to stably control a drone, while meeting the necessary constraints. Moving forward this experience would play an imperative role in implementing flight trajectories that can act as a meaningful and convenient form of communication. Future work should investigate the likelihood of other potential applications for these results.

VIII. Acknowledgments

The group would like to thank Professor Tim Bretl for his help throughout our project. We would also like to thank the teaching assistants of our lab sections Rihan Aaron Dsilva, Nagesh Eranki, Hamid Osooli, Hongri Zhao, and Destiny Fawley for their guidance and assistance throughout the project. We would like thank the whole course staff for their help in building up our understanding of drone dynamics and troubleshooting.

References

[1] Lee, S.-h., Kang, S. H., and Kim, Y., "Trajectory tracking control of quadrotor UAV," *2011 11th International Conference on Control, Automation and Systems*, IEEE, 2011, pp. 281–285.

[2] Hernández-Martínez, E. G., Fernandez-Anaya, G., Ferreira, E. D., Flores-Godoy, J.-J., and Lopez-Gonzalez, A., "Trajectory tracking of a quadcopter UAV with optimal translational control," *IFAC-PapersOnLine*, Vol. 48, No. 19, 2015, pp. 226–231.

[3] Howell, T. A., Cleac'h, S. L., Singh, S., Florence, P., Manchester, Z., and Sindhwani, V., "Trajectory Optimization with Optimization-Based Dynamics," 2023. URL `https://arxiv.org/pdf/2109.04928`, accessed: November 8, 2024.

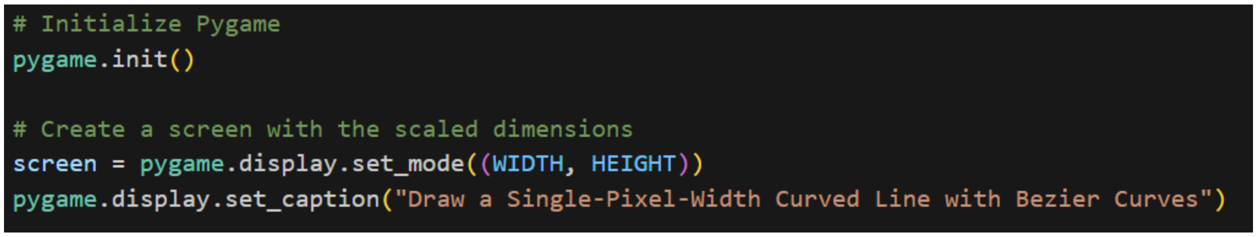
[4] Hubbs, S., Mata, I., McLeod, E., and Williams, J., "Trajectory Computation and Optimization of a Crazyflie 2.0," *AE 483 Final Project Report, University of Illinois at Urbana-Champaign*, 2022.

[5] Petchenik, I., "Filling up the tanks: the Airbus A321XLR 13 hour test flight," , 2022. URL `https://www.flightradar24.com/blog/filling-up-the-tanks-the-airbus-a321xlrs-13-hour-test-flight/`, accessed November 8, 2024.

[6] Andaluz, V., López, E., Manobanda, D., Guamushig, F., Chicaiza, F., Sánchez, J., Rivas, D., Pérez Gutiérrez, M., Morales, V., and Sánchez, C., "Nonlinear Controller of Quadcopters for Agricultural Monitoring," 2015. https://doi.org/10.1007/978-3-319-27857-5_43.

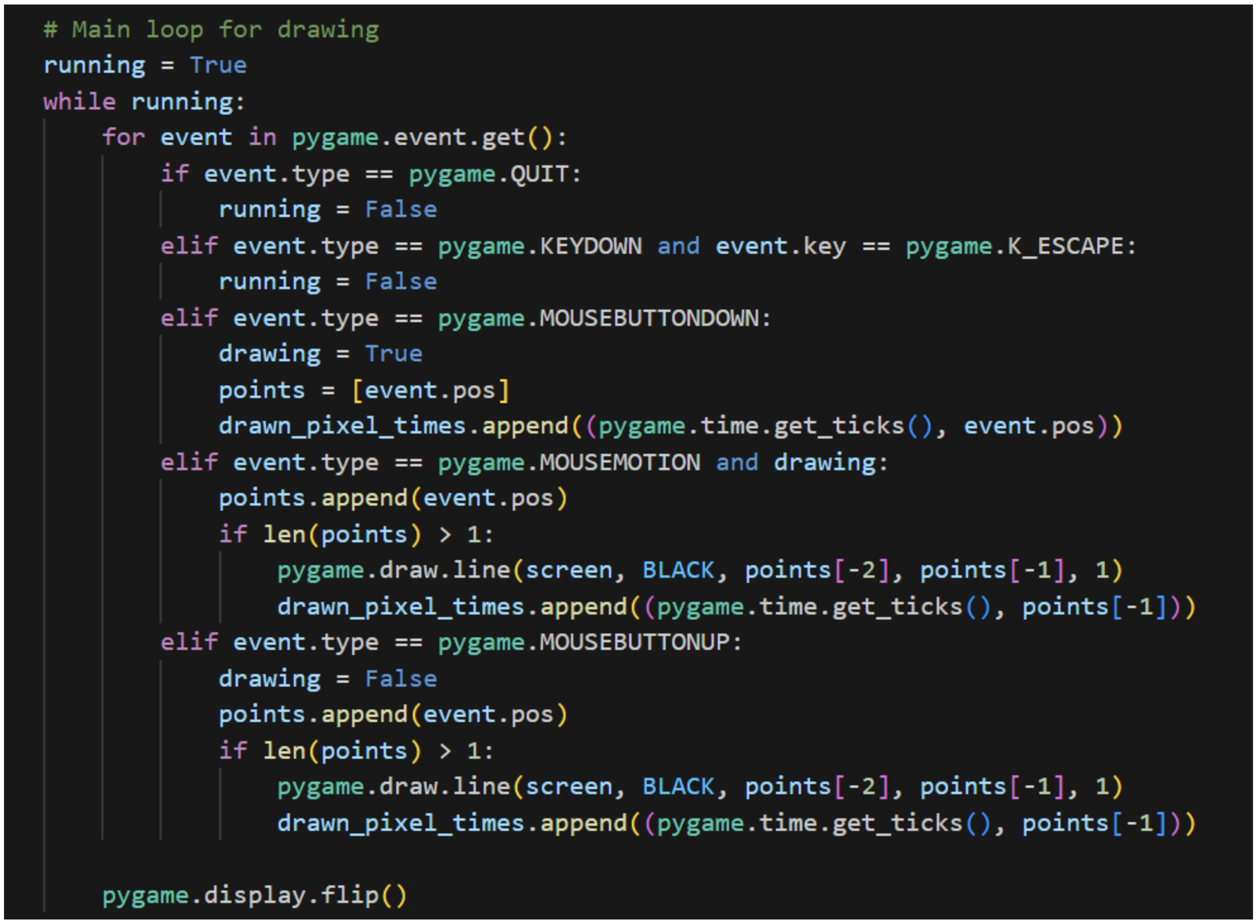
Appendix

Code



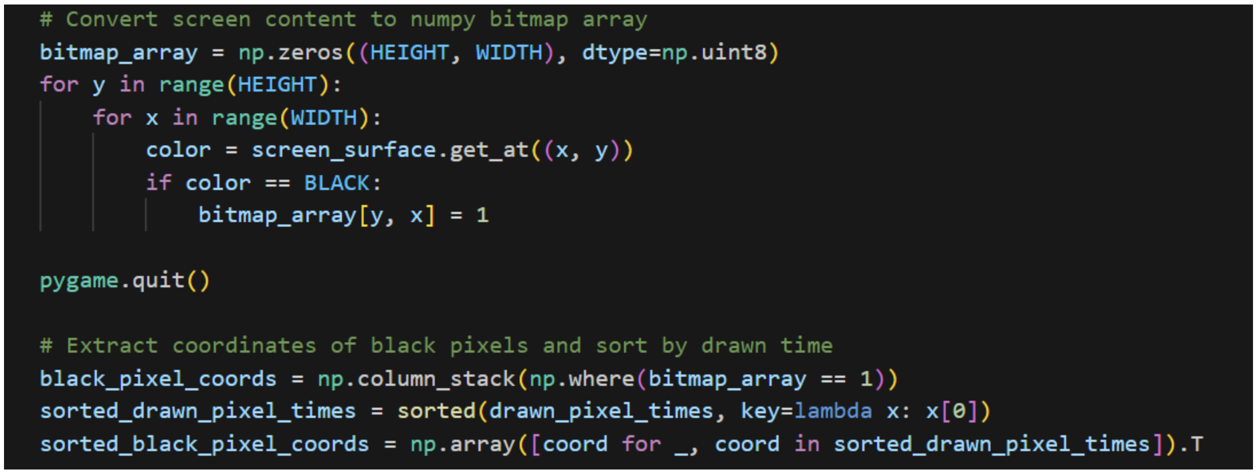
```python
# Initialize Pygame
pygame.init()

# Create a screen with the scaled dimensions
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Draw a Single-Pixel-Width Curved Line with Bezier Curves")
```

Fig. 7 Pygame Initialization



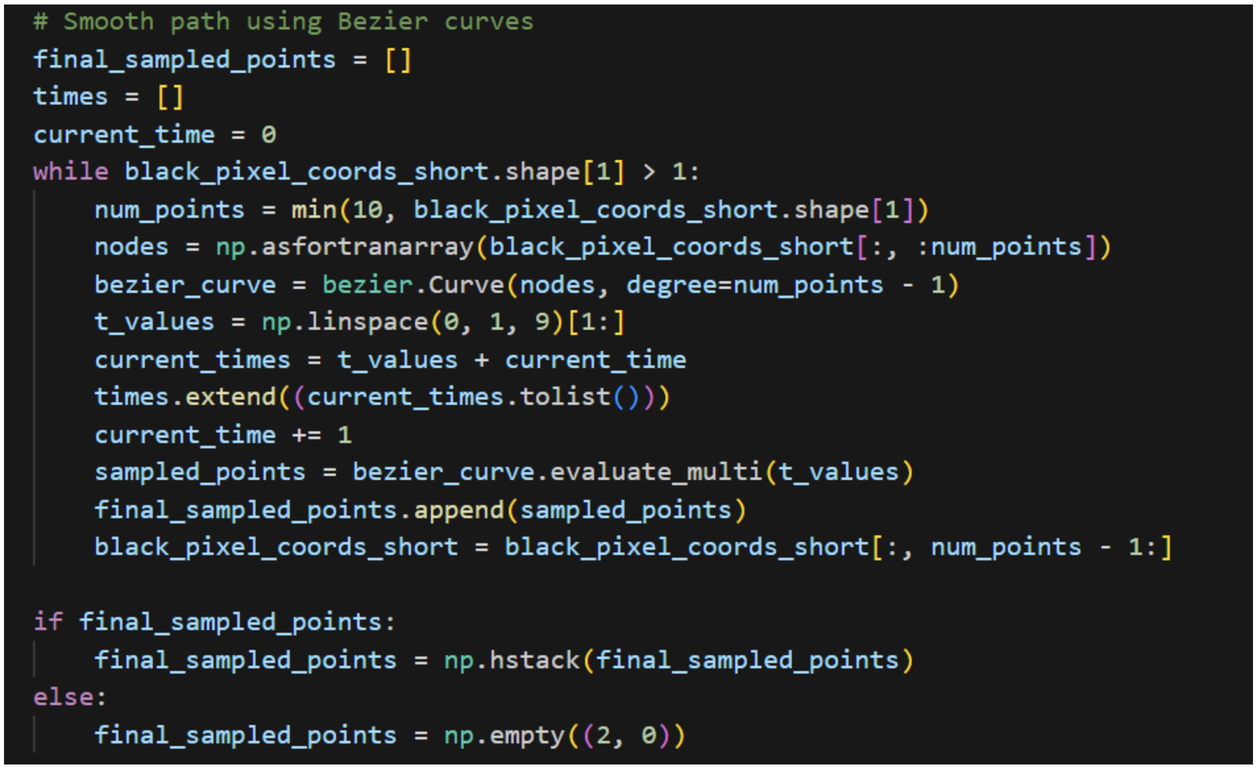
```python
# Main loop for drawing
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            drawing = True
            points = [event.pos]
            drawn_pixel_times.append((pygame.time.get_ticks(), event.pos))
        elif event.type == pygame.MOUSEMOTION and drawing:
            points.append(event.pos)
            if len(points) > 1:
                pygame.draw.line(screen, BLACK, points[-2], points[-1], 1)
                drawn_pixel_times.append((pygame.time.get_ticks(), points[-1]))
        elif event.type == pygame.MOUSEBUTTONUP:
            drawing = False
            points.append(event.pos)
            if len(points) > 1:
                pygame.draw.line(screen, BLACK, points[-2], points[-1], 1)
                drawn_pixel_times.append((pygame.time.get_ticks(), points[-1]))

    pygame.display.flip()
```

Fig. 8 Pygame While Loop



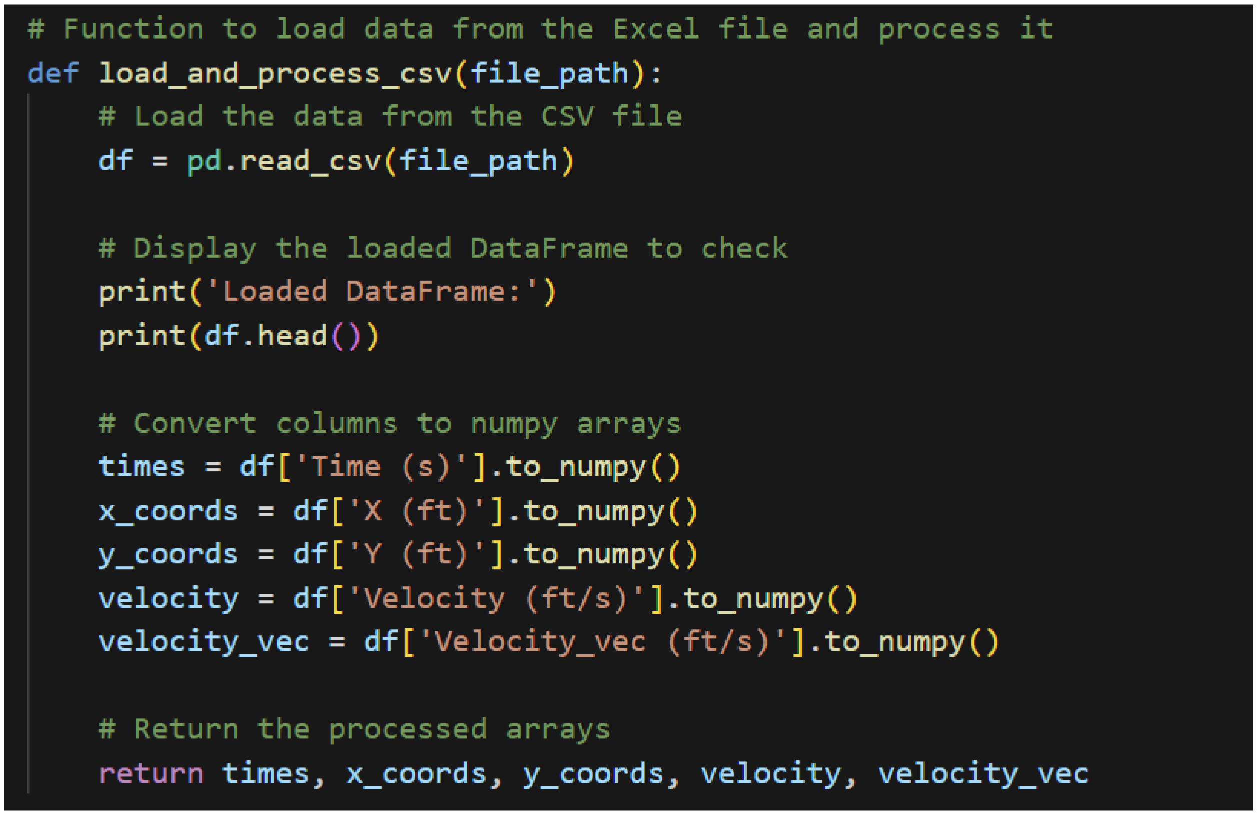
```python
# Convert screen content to numpy bitmap array
bitmap_array = np.zeros((HEIGHT, WIDTH), dtype=np.uint8)
for y in range(HEIGHT):
    for x in range(WIDTH):
        color = screen_surface.get_at((x, y))
        if color == BLACK:
            bitmap_array[y, x] = 1

pygame.quit()

# Extract coordinates of black pixels and sort by drawn time
black_pixel_coords = np.column_stack(np.where(bitmap_array == 1))
sorted_drawn_pixel_times = sorted(drawn_pixel_times, key=lambda x: x[0])
sorted_black_pixel_coords = np.array([coord for _, coord in sorted_drawn_pixel_times]).T
```

Fig. 9 Bitmap Array



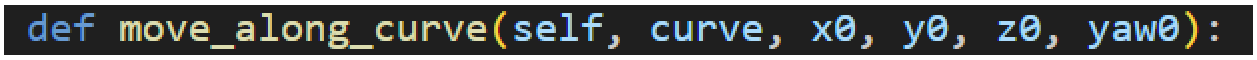
```python
# Smooth path using Bezier curves
final_sampled_points = []
times = []
current_time = 0
while black_pixel_coords_short.shape[1] > 1:
    num_points = min(10, black_pixel_coords_short.shape[1])
    nodes = np.asfortranarray(black_pixel_coords_short[:, :num_points])
    bezier_curve = bezier.Curve(nodes, degree=num_points - 1)
    t_values = np.linspace(0, 1, 9)[1:]
    current_times = t_values + current_time
    times.extend((current_times.tolist()))
    current_time += 1
    sampled_points = bezier_curve.evaluate_multi(t_values)
    final_sampled_points.append(sampled_points)
    black_pixel_coords_short = black_pixel_coords_short[:, num_points - 1:]

if final_sampled_points:
    final_sampled_points = np.hstack(final_sampled_points)
else:
    final_sampled_points = np.empty((2, 0))
```

Fig. 10 Creation of Segmented Bezier Curve



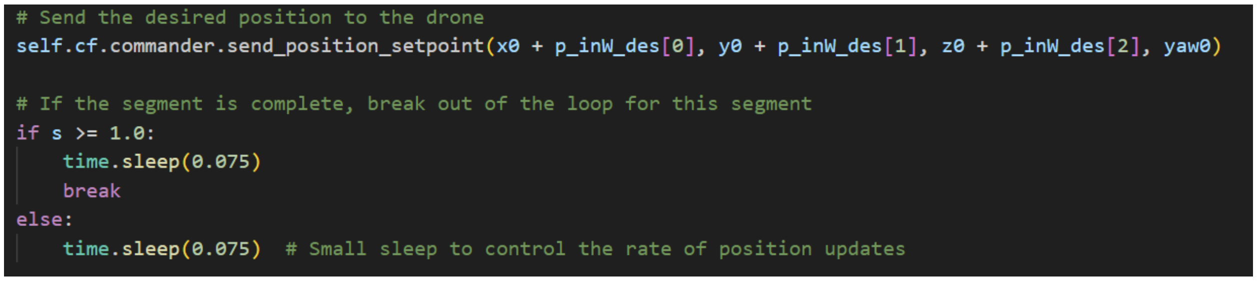
```python
# Function to load data from the Excel file and process it
def load_and_process_csv(file_path):
    # Load the data from the CSV file
    df = pd.read_csv(file_path)

    # Display the loaded DataFrame to check
    print('Loaded DataFrame:')
    print(df.head())

    # Convert columns to numpy arrays
    times = df['Time (s)'].to_numpy()
    x_coords = df['X (ft)'].to_numpy()
    y_coords = df['Y (ft)'].to_numpy()
    velocity = df['Velocity (ft/s)'].to_numpy()
    velocity_vec = df['Velocity_vec (ft/s)'].to_numpy()

    # Return the processed arrays
    return times, x_coords, y_coords, velocity, velocity_vec
```

Fig. 11   Send Trajectory to Excel



```python
def move_along_curve(self, curve, x0, y0, z0, yaw0):
```

Fig. 12   Defining Function



```python
# Send the desired position to the drone
self.cf.commander.send_position_setpoint(x0 + p_inW_des[0], y0 + p_inW_des[1], z0 + p_inW_des[2], yaw0)

# If the segment is complete, break out of the loop for this segment
if s >= 1.0:
    time.sleep(0.075)
    break
else:
    time.sleep(0.075)  # Small sleep to control the rate of position updates
```

Fig. 13   Internal Changes