Project 1 Report: Trajectory Tracking with Quadrotor Control

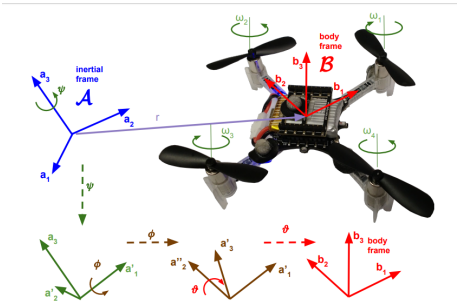
Michael Sobrepera, Jesus Gallegos, and Jeremy Saslaw

***Abstract*— Quadrotors are a key technology in today's robotic and Unmanned Aerial Vehicle (UAV) industries, and thus developing a superior method of flying and controlling them is of a high level of interest. With regards to the controlling of the device, both geometric and non-geometric methods are capable of getting the job done, but in the interest of optimization it was determined that the geometric controller provided the most accurate and efficient method between the two. Furthermore, trajectory planning has numerous different implementations, but it was determined that in order to minimize jerk, a fifth degree polynomial continuous trajectory planner would work best. After tweaking various gains, the CrazyFlie could be controlled to track various generated trajectories efficiently, at high speeds, and to a small degree of error.**

I. INTRODUCTION

The key objective of this project involved adequately flying a Quadrotor. By first studying quad rotor dynamics, applying this theory to a geometric controller, and subsequently creating and integrating a trajectory planning algorithm, a CrazyFlie quad rotor was programmed to fly in the VICON area. Each step along this process involved significant theory, designing of algorithms, and iterative testing, which will be described in the sections below. Notably, while the controller was developed before the trajectory planner, once each task was completed and proven to work sufficiently well through simulation, various parameters and variables were looked back upon and modified once the controller and trajectory planner could be tested together.

II. QUADROTOR DYNAMICS



(a)

Fig. 1: CrazyFlie 2.0

Figure 1 illustrates the coordinate systems for the CrazyFlie 2.0 quadrotor. The fixed robot body frame B is obtained by an euler Z-Y-X transformation from the fixed inertial frame A and a translation of r. The rotation is defined by a rotation of ψ around a_3, then a rotation of ϕ around the new a_1 axis, and finally a rotation of θ around the new a_2 axis. The robot body frame B is fixed at the center of mass of the robot with b_1, b_2, and b_3 defining B.

Using the Euler angles defined, we can produce the rotation matrix ${}^{A}R_{B}$ that takes a given point in B and expresses it in A. We can also relate the angular velocity of the robot to the Euler angle velocities through:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\cos(\phi)\sin(theta) \\ 0 & 1 & \sin(\phi) \\ \sin(\theta) & 0 & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{1}$$

We obtain the equation of motion for the robots center of mass through a force balance on the robots mass, taking into account the force of gravity and the force from each of the motors on the robot. The resulting equation of motion is expressed as:

$$m^{A}\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{A}R_{B} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{2}$$

We chose to express the motion of the center of mass in the fixed inertial frame A. Since the motors' force vectors are aligned with axis b_3, we can multiply their sums by ${}^{A}R_{B}$ to obtain their equivalent representation in the fixed inertial frame.

We obtain the equation of motion for the attitude of the robot by relating the rate of change of the robot's angular momentum to the the moments generated by the motors. We express the equation of motion for the attitude in the body frame B as:

$$I^{B}\ddot{w} = {}^{B}M - {}^{B}w \times I^{B}w \tag{3}$$

For the CrazyFlie 2.0 this equation can be written with the angular velocities p, q, r as:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4}$$

Given the two equations of motion, we can define two inputs related to the motors we can develop controllers around. In the center of mass equation of motion we define:

$$u_1 = F_1 + F_2 + F_3 + F_4 \tag{5}$$

In the attitude equation of motion we define the second input as the vector:

$$\boldsymbol{u_2} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} \tag{6}$$

The two resulting equations of motions can then be rewritten as:

$$m^A\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^A R_B \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} \tag{7}$$

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \boldsymbol{u_2} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{8}$$

Using these equations and these two inputs, we develop a controller to command the robot's position within the fixed inertial reference frame *A*.

III. Geometric Controller

The implemented controller is based on the controller developed in [1]. This controller allows the quadrotor to follow specified trajectories, and it can be broken down into a position and attitude controller.

For the position controller we define vector errors on position and velocity. These are defined as the difference between actual and desired position and velocity vectors:

$$\mathbf{e_p} = \mathbf{r} - \mathbf{r_T}, \mathbf{e_v} = \dot{\mathbf{r}} - \dot{\mathbf{r_T}} \tag{9}$$

The PD position controller can be written as:

$$\ddot{\mathbf{r}}^{\mathbf{des}} = \ddot{\mathbf{r_T}} - \mathbf{K_d}(\mathbf{e_v}) - \mathbf{K_p}(\mathbf{e_d}) \tag{10}$$

Here, $\mathbf{K_d}$ and $\mathbf{K_p}$ are diagonal matrices containing the gains for the x, y, and z errors along the diagonal. The value $\ddot{\mathbf{r}}^{des}$ is the acceleration we hope to command through input u_1. Using the equation of motion for the center of mass, we can determine the value of u_1 in terms of $\ddot{\mathbf{r}}^{des}$. First, we define:

$$\mathbf{c} = {}^A R_B \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = {}^A R_B \mathbf{b_3} \tag{11}$$

Here, $\mathbf{c}$ is the body frame axis $\mathbf{c}$ represented in the fixed inertial frame. Now we can isolate u_1:

$$u_1 = \mathbf{c^T}[m^A\ddot{\mathbf{r}} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}] \tag{12}$$

We can simplify this equation by denoting that the right hand side of the equation is c^T multiplied by the desired force applied on the robot:

$$\mathbf{F}\mathbf{des} = m^A\ddot{\mathbf{r}} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{13}$$

We use geometric intuition to begin building the geometric controller. In particular, we know that the robot's motors can only exert a force along the body axis $\mathbf{b_3}$. Thus, we seek to use the attitude controller to align axis $\mathbf{b_3}$ with the desired thrust $\mathbf{F}\mathbf{des}$. Using this, we can define a new triad $R_{des} = [\mathbf{b_1^{des}}; \mathbf{b_2^{des}}; \mathbf{b_3^{des}}]$. We also chose to align $\mathbf{b_1^{des}}$ with the desired yaw angle. This means that $\mathbf{b_2^{des}}$ must be perpendicular to both $\mathbf{b_3^{des}}$ and the desired yaw vector a_ψ. The yaw vector lies in the (a_1, a_2) plane, and is defined as:

$$a_\psi = \begin{bmatrix} \cos a_\psi \\ \sin a_\psi \\ 0 \end{bmatrix} \tag{14}$$

Now we can define the three vectors denoting the desired orientation as:

$$\mathbf{a_3^{des}} = \frac{\mathbf{F^{des}}}{||\mathbf{F^{des}}||} \tag{15}$$

$$\mathbf{a_2^{des}} = \frac{\mathbf{b_3^{des}} \times \mathbf{a}_\psi}{||\mathbf{b_3^{des}} \times \mathbf{a}_\psi||} \tag{16}$$

$$\mathbf{a_1^{des}} = \mathbf{b_2^{des}} \times \mathbf{b_3^{des}} \tag{17}$$

With R_{des}, we can define orientation and angular velocity errors we can use to define a control law. In particular, we use the orientation error vector proposed in [1]:

$$\mathbf{e_R} = \frac{1}{2}(R_{des}^T {}^A R_B - {}^A R_B^T R^{des})^v \tag{18}$$

The v operator is a map from so(3) to $\mathbb{R}^3$. For a given 3x3 matrix R, it returns:

$$R^v = \begin{bmatrix} -R(2,3) \\ R(1,3) \\ -R(1,2) \end{bmatrix} \tag{19}$$

The angular velocity error is simply:

$$\mathbf{e_w} = \mathbf{w} - \mathbf{w^{des}} \tag{20}$$

In our application, we select w^{des} to be zero, so the error simply becomes:

$$\mathbf{e_w} = \mathbf{w} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{21}$$

The value of $\mathbf{e_R}$ decreases to zero when $R_{des} = {}^A R_B$. Thus, if we apply a torque in the direction if the error, it will decrease. The applied torque will be the final control input $\mathbf{u_2}$:

$$\mathbf{u_2} = I(-K_R\mathbf{e_r} - K_w\mathbf{e_w}) \tag{22}$$

In this equation *I* is the moment of inertia for the robot and K_R and K_w are the diagonal gain matrices.

Equations (12) and (22) encompass the control law that govern the position and attitude controller for our robot.

IV. Controller Gains and Adjustments

In total, our controller used 4 gain matrices for the position and attitude control laws. The position controller used a proportional and a derivative gain matrix:

$$K_p = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 15 \end{bmatrix} \tag{23}$$

$$K_d = 0.7K_p \tag{24}$$

Given that the quadrotor is rotationally symmetric in the x-y, or $(a_1, a2)$ plane, it makes sense that the x and y position gains are the same value. We can easily imagine that if the b_1 vector begins a path by pointing in the a_1 direction, and rotates by a yaw angle of 90 degrees, it will now be pointing in the a_2 direction. Thus, the x and y position gains should be the same.

The gains for the geometric controller were:

$$K_R = \begin{bmatrix} 180 & 0 & 0 \\ 0 & 180 & 0 \\ 0 & 0 & 180 \end{bmatrix} \tag{25}$$

$$K_w = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix} \tag{26}$$

Our controller also makes use of a slight adjustment to the gravity parameter of our robot. During initial testing, we commanded the robot to hover at a steady height. However, the observed hover height was lower than the desired hover height by a fixed offset. To mitigate this, we made an adjustment to the gravitational force on the robot, ultimately multiplying its value by 1.12. In other words, we adjusted equation (13) as follows:

$$\mathbf{Fdes} = m^A\ddot{\mathbf{r}} + 1.12 \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{27}$$

All flight tests presented inn subsequent sections use these control gains and adjustments.

V. Trajectory Planner

For trajectory planning, we implemented a 5th degree polynomial continuous trajectory planner. This planner minimizes jerk.

The first step involves clearing any trajectory points which are sampling artifacts, lying on a line at consistent distances from each other. In doing this, the planner will be able to avoid hitting redundant points. In practice, this is implemented by removing points one index past where all of the differentials of the differentials of the trajectory points are equal to zero. This allows the end points to remain, removing the middle points only.

The next step is to determine the time to attempt to hit each point. The first point is taken at time zero. The length of each trajectory segment is then taken as the log of the distance between each point and its prior point plus one, multiplied by one over the speed scaling factor, defined here as k_t

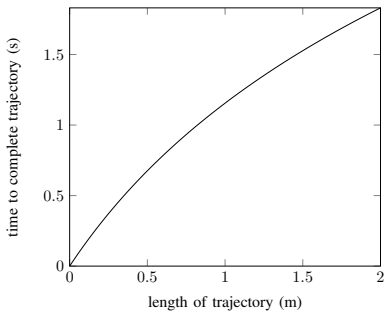


Fig. 2: A graph showing the time for each segment based on the distance between points, using our scaling factor of $k_t = 0.6$

$$t_{seg} = \ln\left(\|x_{i-1} - x_i\| + 1\right) * \frac{1}{k_t} \tag{28}$$

In our testing, we found a value of $k_t = 0.6$ to work well. This segment time function yields linear increases in time near zero distance between points with a distance per time of k_t. The time increase relative to higher distances then falls off in a logorithmicly. This allows small motions where more time is needed in order to handle acceleration that must be handled, while still enabling long distance fast motions to occur. This can be seen in fig. 2

The trajectories themselves are defined as a polynomial in three space for each trajectory segment. All segments are solved over simultaneously to generate the trajectory coefficients by solving a matrix equation which sets the position, velocity, and acceleration of the end of each trajectory segment equal to the start of the next one, with the time at each intersection defined as above and positions given. For example, for a simpler 3rd degree polynomial with one intermediate waypoint, this would look like the equation shown in section V.

Having then solved for the coefficients, in as many dimensions as needed, for as many points as needed, when the robot queries for the desired position, acceleration, and velocity, the input time is compared to the times found above to determine the current segment. The coefficients for that segment are then used to calculate the exact desired kinematics at that time.

The system which we used is actually general enough to work for any specified odd degree polynomial and for both continuous and non-continuous smooth and continuous non-smooth piece-wise trajectories.

VI. Results

The results of our combination of trajectory generator and controller produced robust, fast, and smooth flight. Initial tuning was needed to determine the maximum value which could be used for k_t and for the constants in the PID controller on thrust of the vehicle. Summary results can be seen in table I.

$$\begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3t_0^2 & 2t_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6t_0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t_m^3 & t_m^2 & t_m & 1 & -t_m^3 & -t_m^2 & -t_m & 1 & 0 & 0 & 0 & 0 \\ 3t_m^2 & 2t_m & 1 & 0 & -3t_m^2 & -2t_m & -1 & 0 & 0 & 0 & 0 & 0 \\ 6t_m & 2 & 0 & 0 & -6t_m & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_m^3 & t_m^2 & t_m & 1 & -t_f^3 & -t_f^2 & -t_f & 1 \\ 0 & 0 & 0 & 0 & 3t_m^2 & 2t_m & 1 & 0 & -3t_f^2 & -2t_f & -1 & 0 \\ 0 & 0 & 0 & 0 & 6t_m & 2 & 0 & 0 & -6t_f & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_f^3 & t_f^2 & -t_f & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3t_f^2 & 2t_f & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6t_f & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ a_m \\ b_m \\ c_m \\ a_f \\ b_f \\ c_f \end{bmatrix} = \begin{bmatrix} x_0 \\ 0 \\ 0 \\ x_m \\ 0 \\ 0 \\ x_f \\ 0 \\ 0 \end{bmatrix}$$

Fig. 3: An example of the matrix equation which would be solved to generate coefficients of the trajectory for smooth, 3rd degree polynomial motion. There is an initial, middle, and final point. Note that the coefficient values x values can be of any dimension needed, normally 4 for x,y,z,yaw

Dataset	Distance [m]	Planned distance [m]	Time [s]	Velocity [m/s]
Path 1	5.4624	5.0488	7.1092	0.7684
Path 2	10.3203	9.7795	12.3903	0.8329
Path 3	5.6247	4.8767	9.8500	0.5710

TABLE I: A summary of results over the three paths.

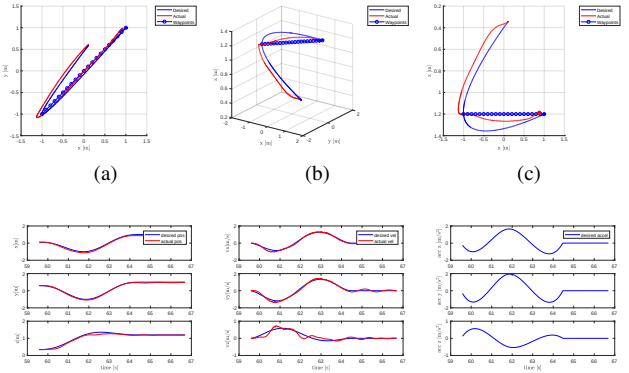


Fig. 4: Path 1: **a)** the path through space in the xy plane **b)** the path through space in an isometric view **c)** the path trough space in the xz plane **d)** the position e) the velocity **e)** the acceleration. In all plots, blue is desired values and red is actual values.

As can be seen in fig. 4, the first step of the trajectory generator removes the points in the middle of the line of the first trajectory, which are assumed to be redundant samples from a higher level planner. The vehicle then moves quickly and smoothly through the two trajectory segments. However, if it were important that the vehicle travel through the intermediate points, our trajectory would have been non-ideal.

For the second path, shown in fig. 5, the vehicle takes a very loopy trajectory, successfully hitting all points at high speed. However, to achieve its trajectory, the vehicle deviates from the waypoint envelope quite dramatically in the z direction. For our situation, this worked well, however, if there was a ceiling on the flight space, that could cause problems. It is thus noted that depending on the exact application for a given quad rotor, each trajectory planner will have its own strengths and weaknesses, which need to be tuned towards the given task at hand.

For the third path (fig. 6), the vehicle again successfully hits all waypoints at high speed. Due to the spacing of the points, the trajectory does not this time exit the envelope of the waypoints to a significant degree, making the trajectory likely very safe for this waypoint set.

In summary, our trajectory tracking with geometric control worked well once tuned. Looking at the position plots, the results are quite good. There are some oscillations in velocity, however they don't appear to affect overall performance significantly. We did find that we were pushing the vehicle's motors to the point of saturation. Additionally, as the battery died, the vehicle became unable to handle sharp turns. For example, the vehicle would be incapable of maintaining the trajectory near the first waypoint in the third path (fig. 6), would become unstable, and crash. This highlights a fundamental flaw in this design, which is at the trajectory level open loop. If a vehicle is unable to maintain the trajectory, the results are far from optimal or safe. Most of these problems could be solved by being less aggressive, primarily with the time constant used in the trajectory planning, but also with the the controller gains.

It is thus concluded, as expected, that there is a natural give and take between speed of completing a trajectory versus the actual accuracy of hitting each way-point. The end goal would always be to optimize both characteristics, but in

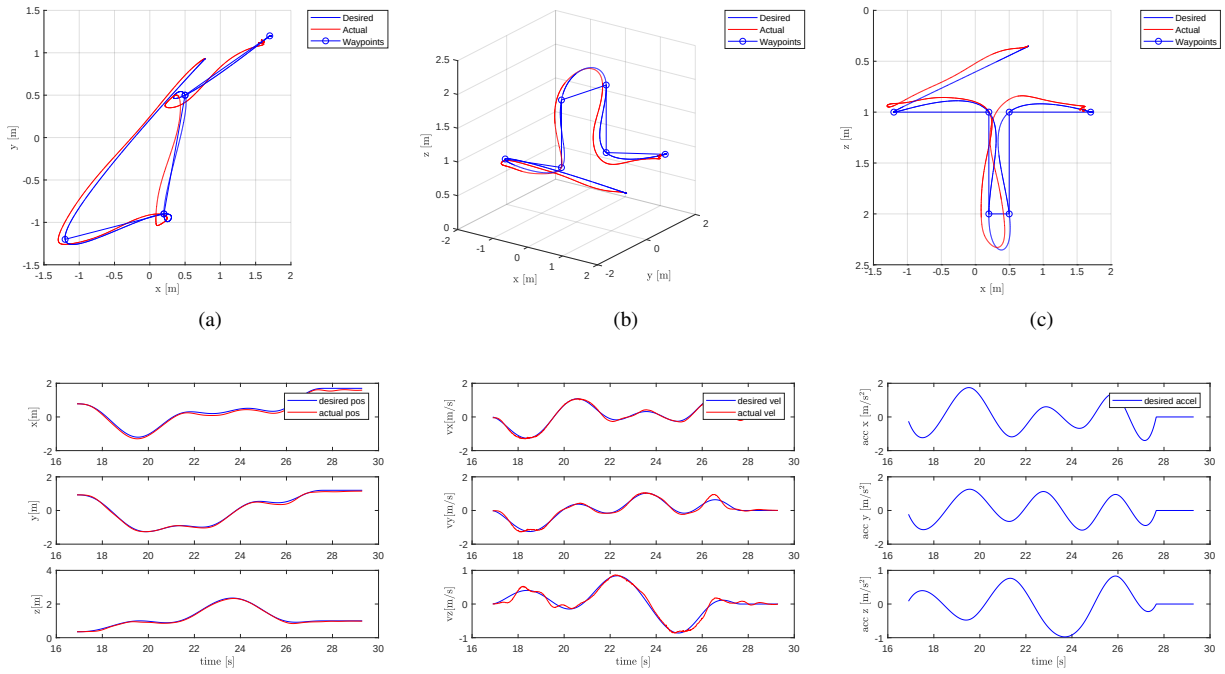


Fig. 5: Path 2: **a)** the path through space in the xy plane **b)** the path through space in an isometric view **c)** the path trough space in the xz plane **d)** the position e) the velocity **e)** the acceleration. In all plots, blue is desired values and red is actual values.

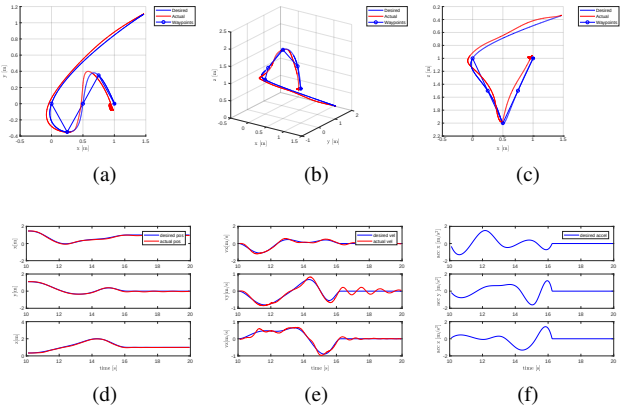


Fig. 6: Path 3: **a)** the path through space in the xy plane **b)** the path through space in an isometric view **c)** the path trough space in the xz plane **d)** the position e) the velocity **e)** the acceleration. In all plots, blue is desired values and red is actual values.

the end a given Quadrotor should be optimized towards its specific application.

REFERENCES

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Shanghai, China, May 2011.