2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Near minimum-time trajectories for quadrotor UAVs in complex environments

Jonathan Jamieson[1] and James Biggs[2]

Abstract— **This paper presents a heuristic framework that generates trajectories for quadrotor UAVs in complex environments that smoothly follow a series of waypoints with desired boundary states on the derivatives. The waypoints can be chosen manually or found using a sampling-based path planner such as RRT for complex environments with obstacles. A trajectory in the virtual domain is found using polynomials parametrised by an abstract argument that are numerically optimised to give the shortest geometrical path length. A mapping function is used to ensure the kinodynamic limits on the velocity and acceleration throughout the trajectory are satisfied. The mapping function is found heuristically with an algorithm that minimises the trajectory time.**

I. INTRODUCTION

Unmanned aerial vehicles are capable of autonomously performing a variety of tasks such as remote sensing, aerial photography and environmental monitoring. Motion planning is necessary to guide vehicles through their environment and reach places of interest. Complex environments with obstacles require more sophisticated planning than point-to-point trajectories. Even finding a feasible, let alone optimal trajectory, can be a challenging problem. Motion planning has been shown to be NP-hard in all but the simplest cases [1]. In [2] they propose using a sampling-based planner to find the waypoints of a geometrical path before using them to define a trajectory as a function of time that a quadrotor can track. The planner they use is part of the Rapidly-Exploring Random Tree (RRT) family with the modifications suggested by [3] for optimality. In higher dimension systems sampling-based methods make the problem more tractable.

The differential flatness of quadrotor dynamics [4] also simplifies the planning problem for these vehicles by allowing the translational axes in the inertial frame to be decoupled from the rotational path planning, computed independently and recombined to define the trajectory. Analytical basis functions are used to describe the motion and their coefficients chosen to satisfy conditions placed on the trajectory. These conditions are boundary constraints on the position and higher order derivatives and waypoints that must be passed during the motion. In this paper power series polynomials are used but other basis functions such as Bezier curves, B-splines and Legendre Polynomials could equally be applied.

[1]Jonathan Jamieson is with the Department of Mechanical & Aerospace Engineering, University of Strathclyde, Glasgow, United Kingdom jonathan.jamieson@strath.ac.uk

[2]James Biggs is with the Dipartimento di Scienze e Tecnologie Aerospaziali, Politecnico di Milano, Milano, Italy jamesdouglas.biggs@polimi.it

Perhaps the simplest approach is to use fifth-order polynomials to control the velocity and acceleration for a point-to-point manoeuvre [5]. The computational time is negligible because the problem can be written as a linear equation, however, the final time needs to be found through an iterative process. To ensure the dynamics throughout the entire trajectory are feasible an excessively large final time is often required when only a segment of the trajectory is pushing the kinodynamic limits. Increasing the order of the function gives better flexibility to enhance the performance but there will no longer be a unique solution. Instead a solution can be found that minimises some cost function, such as [2] where the problem is formulated as an unconstrained quadratic program where the coefficients are found to minimise snap. Again, the dynamic limits are met by changing the final time until a suitable value is found.

If the order of the function is too high numerical stability can be affected, such as when greater control of the derivatives is desired during the trajectory. One option is to separate the temporal information from the geometrical path. A mapping function is used to relate an abstract argument that parametrises the function that describes the position to the time domain. This method has been used since the 1980s [6] and is known by a variety of names including adjusting the velocity profile, planning in the virtual domain and time scaling. A variety of systems and problems have been investigated using this method such as the re-orientation of spacecraft [7], [8] and robotic arms [9]. A point-to-point quadrotor virtual trajectory generation method was presented in [10].

We present a heuristic approach to finding feasible, near minimum-time trajectories in complex environments. As in [2] a sampling-based planning method is used to generate the waypoints for the trajectory to follow. However, we attempt to minimised trajectory time and plan on the virtual domain before mapping to the time domain. Our method for mapping to the time domain is unlike [10] because it allows for trajectories with non-zero boundary states. Also our formulation allows for the ability to pass through multiple waypoints, enabling more complex trajectories to be found. Section II explains the formulation of polynomials parametrised in the virtual domain with suitable boundary conditions that form a minimum length trajectory that passes through the specified waypoints. The method for mapping into the time domain is in Section III. Two cases are used to demonstrate our framework in Section IV. The first is a short multi-waypoint planar trajectory and the second uses the output from a sampling-based path planning algorithm

978-1-5090-3762-9/16/$31.00 ©2016 IEEE

to motion plan in a 3D environment with obstacles. Finally, our conclusions and further work can be found in Section V.

II. Virtual Domain Polynomial Trajectories

In this section we begin by defining what is required for a quadrotor reference trajectory given the differential flatness of the system. We then develop a method for interpolating the waypoints subject to boundary constraints with polynomials parametrised by an abstract argument $\tau \in [0, 1]$ for each axis in the virtual domain.

A. Quadrotor differential flatness

Quadrotor dynamics have been well studied and covered previously [11]. A quadrotor has four motor outputs and has six degrees of freedom (3 translational coordinates and 3 rotations) so it is under-actuated. However, since quadrotors have been shown to be differentially flat [4] motion planning has been greatly simplified. An expression for the outputs can be given if the four inputs x_1, x_2, x_3 and ψ and their derivatives are known. The first three are the Cartesian coordinates of the quadrotor in the inertial frame and ψ is the yaw angle which is normally (but not required to be) fixed at zero for theoretical trajectory planning. Provided the four inputs are differentiable and remain within the kinodynamic limits the trajectory will be feasible.

B. Defining the virtual trajectories

Power series polynomials have been used as basis functions in motion planning algorithms for autonomous systems previously in [5], [12], [13] as they are simple to manipulate, differentiate and implement. Other basis functions have been used for similar problems, including Legendre Polynomials [14] and B-splines [15] which in some cases offer greater numerical stability than standard polynomials. The position on each translational axis in the inertial frame is described by a polynomial:

$$P_a(\tau) = p_{a,n}\tau^{d_a} + p_{a,n-1}\tau^{d_a-1} + \cdots + p_{\tau,0} = \sum_{n=0}^{d_a} p_{a,n}\tau^n \tag{1}$$

where $p_{a,n}$ are the coefficients, d_a is the degree of the polynomial and a is the axis number. The number of axes depends on the problem but the method remains the same because of the decoupling. The coordinates of each waypoint is applied to the given axis using:

$$P_a(\tau_w) = x_{a,w} \tag{2}$$

where w is the waypoint number. Each waypoint has an associated value τ_w in the virtual domain. The waypoints at the beginning and end of the trajectory will have a τ_w value of $\tau_i = 0$ and $\tau_f = 1$ respectively. Any waypoints between are referred to as intermediate waypoints and their τ_w value is found using the method in the next section. The conditions on the derivatives at the boundary values are $P_a^{(r)}(\tau)$ where r is the r^{th} derivative. The boundary derivatives on the virtual domain do not necessarily equal the boundary derivatives required on the trajectory on the time domain. Section III explains how to find the correct values on the virtual domain given the boundary values of the mapping function.

In this section it is assumed τ_w for each waypoint is known using the method described in Section II-C. The degree of the polynomial that describes the motion along a given axis should be such that when all the necessary conditions are applied a linear equation can be formed. The constraints on the polynomials can be formulated as a linear equation:

$$M_a\mathbf{p_a} = \mathbf{c_a} \tag{3}$$

where $M_a \in \mathbb{R}^{k_a \times k_a}$ is a matrix where each row applies either a boundary constraint or waypoint condition to the trajectory and k_a is number of conditions applied ($k_a = d_a + 1$). The vectors containing the polynomial coefficients and the values of the conditions are $\mathbf{c_a} \in \mathbb{R}^{k_a}$ and $\mathbf{p_a} \in \mathbb{R}^{k_a}$ respectively. If for example, a trajectory had an axis with boundary positions and velocities, and two waypoints then $k_a = 6$. The polynomial coefficients in linear equation (3) can be solved using Gaussian Elimination with functions like Matlab's $mldivide$ which is more robust than computing the inverse $\mathbf{p_a} = M_a^{-1}\mathbf{c_a}$. After solving for all the axes a path is described in the virtual domain that passes through the waypoints and satisfies the conditions applied at the endpoints.

C. Optimising the path length

The intermediate waypoints are those between the first and last waypoint. Their positions on the virtual domain form the optimisation vector $\xi = [\tau_{i+1}, \ldots, \tau_{f-1}]^T$ which contains τ_w for each intermediate waypoint. The optimisation vector that minimises the geometrical path length for a given set of waypoints and boundary derivatives is found using a numerical optimiser. For a given optimisation vector there is a unique solution found by solving (3) for each axis. Ideally, the optimisation vector that minimises the path length cost function is found, however, it is not critical because any valid choice of ξ will produce a trajectory that satisfies the waypoint and boundary derivative conditions. A poor choice of ξ will result in an unnecessarily long path. In this paper the path length is minimised which is equivalent to minimising:

$$J(\xi) = \int_0^1 \sum_{a=1}^{A} \dot{P}_a(\tau)^2 \; d\tau \tag{4}$$

where A is the number of axes. Since P_a are polynomials, an analytical solution to the cost function can be found which is less computationally intensive and more accurate than integrating numerically. To ensure the waypoints are followed in the correct order the following must be true:

$$\tau_i < \tau_{i+1} < \tau_2 < \cdots < \tau_{f-1} < \tau_f \tag{5}$$

Equation (5) can be written as a linear constraint:

$$L_a\xi \leq \mathbf{b_a} \tag{6}$$

where L_a is a matrix of the τ_w terms and $\mathbf{b_a}$ is a column vector of the inequality constraints given in (5). The minimisation problem of finding the optimisation vector that minimises the path length can be formulated as:

$$\begin{aligned} \underset{\xi}{\text{minimize}} \quad & J(\xi) \\ \text{subject to} \quad & L_a\xi \leq \mathbf{b_a} \end{aligned} \tag{7}$$

This optimisation problem can be solved numerically using standard tools such as Matlab's $fmincon$ or $patternsearch$. The initial guess can be linearly spaced or the optimisation vector found from a previous solution. As mentioned previously, any choice of the optimisation vector subject to (6) will produce a path that satisfies the constraints even if the allowable computational time has been exceeded. That is, if convergence fails the best available but sub-optimal solution can be used. It should be mentioned that curve fitting using Gauss-Jordan Elimination is not suitable for this problem because the function describing the position along an axis, in our case polynomials, would be parametrised by the position on the other axes. Therefore it would not be possible to apply boundary conditions on the derivatives.

III. Mapping to the time domain

The virtual trajectories parametrised by τ must be converted to the time domain t using a mapping function. The geometrical shape and length of the path remain unchanged after the conversion. However, the derivatives of the position such as velocity and acceleration are changed and if a suitable transformation to the time domain is applied, a feasible, quasi-optimal trajectory can be obtained. The new trajectory will pass through the desired waypoints and match the derivatives required at the boundaries. The algorithm described in Section III-B to find the trajectory in the time domain is iterative in nature. This is because a suitable virtual domain path with acceptable boundary conditions is necessary to generate a usable trajectory as described in Section III-A. If either the boundary mapping values are changed a new virtual trajectory must be found, changes to mapping values between the boundaries does not require replanning the virtual trajectory.

In this paper we chose to find a feasible trajectory whilst algorithmically attempting to minimise the trajectory time. When the problem is formulated this way the final time does not need to be known in advance, that is, no guess needs to be given to the solver. Minimising another performance measure given a fixed time is left as future work.

A. Defining the mapping function

A function that maps the virtual domain τ to the real time domain t is defined as:

$$\lambda(\tau) = \frac{dt}{d\tau} \tag{8}$$

or equivalently:

$$t(\tau) = \int_0^\tau \lambda(\tau)\, d\tau \tag{9}$$

Using the mapping function $\lambda(\tau)$ the position in the virtual domain can be rewritten as a function in the time domain:

$$x_a\left(t(\tau)\right) = P_a(\tau) \tag{10}$$

The function that describes the position in the time domain (10) is not a polynomial (as it was in the virtual domain), however, it is still an analytically defined function that is smooth and continuously differentiable C^∞. After applying the chain rule the first derivative of the position with respect to time (velocity) can be expressed as:

$$\dot{x}_a(t(\tau)) = \frac{dP_a(\tau)}{d\tau}\frac{1}{\lambda(\tau)} \tag{11}$$

Similarly, the acceleration in the time domain is given by:

$$\ddot{x}_a(t(\tau)) = \frac{d^2P_a(\tau)}{d\tau^2}\frac{1}{\lambda(\tau)^2} - \frac{dP_a(\tau)}{d\tau}\frac{d\lambda}{d\tau}\frac{1}{\lambda(\tau)^3} \tag{12}$$

Further derivatives such as jerk can be found by continuing to differentiate but in this paper acceleration is the highest derivative used. A valid mapping function must satisfy $\lambda(\tau) > 0$ because time increases monotonically. Additionally, the derivatives of the endpoints of the virtual trajectory must match the required values in the time domain after the trajectory has been mapped. For the velocity, the initial and final value in the virtual domain are:

$$\begin{aligned} \tfrac{dP_a(\tau_i)}{d\tau} &= \dot{x}_a(0)\lambda(\tau_i), \\ \tfrac{dP_a(\tau_f)}{d\tau} &= \dot{x}_a(T)\lambda(\tau_f) \end{aligned} \tag{13}$$

Likewise, the boundary conditions in the virtual domain for acceleration are:

$$\begin{aligned} \tfrac{d^2P_a(\tau_i)}{d\tau^2} &= \ddot{x}_a(0)\lambda(\tau_i)^2 + \tfrac{dP_a(\tau_i)}{d\tau}\tfrac{d\lambda(\tau_i)}{d\tau}\tfrac{1}{\lambda(\tau_i)}, \\ \tfrac{d^2P_a(\tau_f)}{d\tau^2} &= \ddot{x}_a(T)\lambda(\tau_f)^2 + \tfrac{dP_a(\tau_f)}{d\tau}\tfrac{d\lambda(\tau_f)}{d\tau}\tfrac{1}{\lambda(\tau_f)} \end{aligned} \tag{14}$$

When there are non-zero derivatives at the boundary conditions having $\lambda(\tau) = 1$ may mean it is impossible to find a mapping function that produces a kinodynamically feasible trajectory in the time domain. This is because of the conflict that can exist when trying to keep both the acceleration and velocity magnitudes within the required bounds. To increase $\lambda(\tau)$ to a sufficiently high value required to keep the velocity feasible, $\frac{d\lambda(\tau_i)}{d\tau}$ must also be large enough. However, this causes the second term in (12) to increase until the acceleration is not feasible. To prevent this from happening, the boundary values of the mapping function must start at sufficiently high values which are found iteratively by the mapping algorithm.

B. Finding the mapping function

The optimum mapping function $\lambda(\tau)$ is one that converts the trajectory found in the virtual domain to the shortest time trajectory that is still feasible. The mapping function should be be continuous to ensure the trajectory is smooth with no discontinuities. Another way of thinking about the optimum mapping function is that it should minimise the area

under the τ–$\lambda(\tau)$ graph. It is important to note that even if the optimum mapping function is found, it is only so for that particular geometrical trajectory because the spatial and temporal are considered separately. However, this sacrifice allows the motion planning problem to be more tractable.

The polynomial-based mapping function parametrised by τ presented in this paper is found algorithmically. After placing conditions upon the boundaries and the "mapping nodes" the coefficients are found by solving a linear equation in a similar way to Section II-B. There are three fixed conditions that are unchanged during the process of finding the mapping function:

$$\tau(0) = \dot{\lambda}(0) = \dot{\lambda}(1) = 0 \tag{15}$$

The other conditions are all values of $\lambda(\tau)$ found at the mapping nodes by the algorithm. The mapping nodes are any position on τ where a mapping value is applied. These are positioned using Legendre-Gauss-Lobatto (LGL) node spacing [16] transformed from $[-1, 1]$ to the virtual domain $[0, 1]$. LGL nodes were used because it was observed that near the boundaries the greatest level of control over the mapping function was required. Instead of uniformly increasing the resolution of the mapping nodes across the virtual domain, LGL node spacing has a higher density near the boundaries.

Fig. 1 is a flowchart of the mapping algorithm used to find the mapping values at the mapping nodes. There are three distinct phases (marked by the dashed lines) to the algorithm: 1. Boundary Value Finding, 2. Intermediate Value Finding and 3. Value Refinement. The first phase finds the mapping values at the first, second, penultimate and last mapping node. This is done by fixing the boundary nodes (first and last) whilst gradually increasing the "internal" nodes (second and penultimate). When the difference between an internal node and its corresponding boundary node exceeds a certain value the boundary value is increased and the internal node reset to equal that value. The feasibility of the first and second half of the trajectory is checked independently. When both are acceptable the first phase is over, if only one is acceptable the other half is changed. During this phase the virtual domain path will need to be recalculated whenever the first or last mapping value is changed. The number of times this recalculation is performed should be minimised because it's the most computationally expensive part of the algorithm. This is also why both the first and last mapping nodes are found during the first phase. After finding suitable values for these four nodes, the trajectory can be optimised without regular changes to the virtual domain path. Once the first phase is complete there will always be a feasible trajectory to fall back on.

The second phase finds the mapping values between the second and penultimate node. This is done by considering one node at a time and attempting to reduce its value. When the node cannot be reduced any further without making the trajectory infeasible the algorithm moves to the next node and starts reducing it. The order that the nodes are processed alternates sides from the second node (*LEFT*) to the penultimate node (*RIGHT*). Once all the mapping nodes have values the second phase is complete. The third and final phase (Value Refinement) goes through the mapping values and reduces them until no more reductions can be made. This is necessary because the surrounding mapping nodes values change it may be possible to reduce a particular mapping node further than would have previously been allowable. As the nodes are reduced neighbouring node mapping values may in turn also be improved.

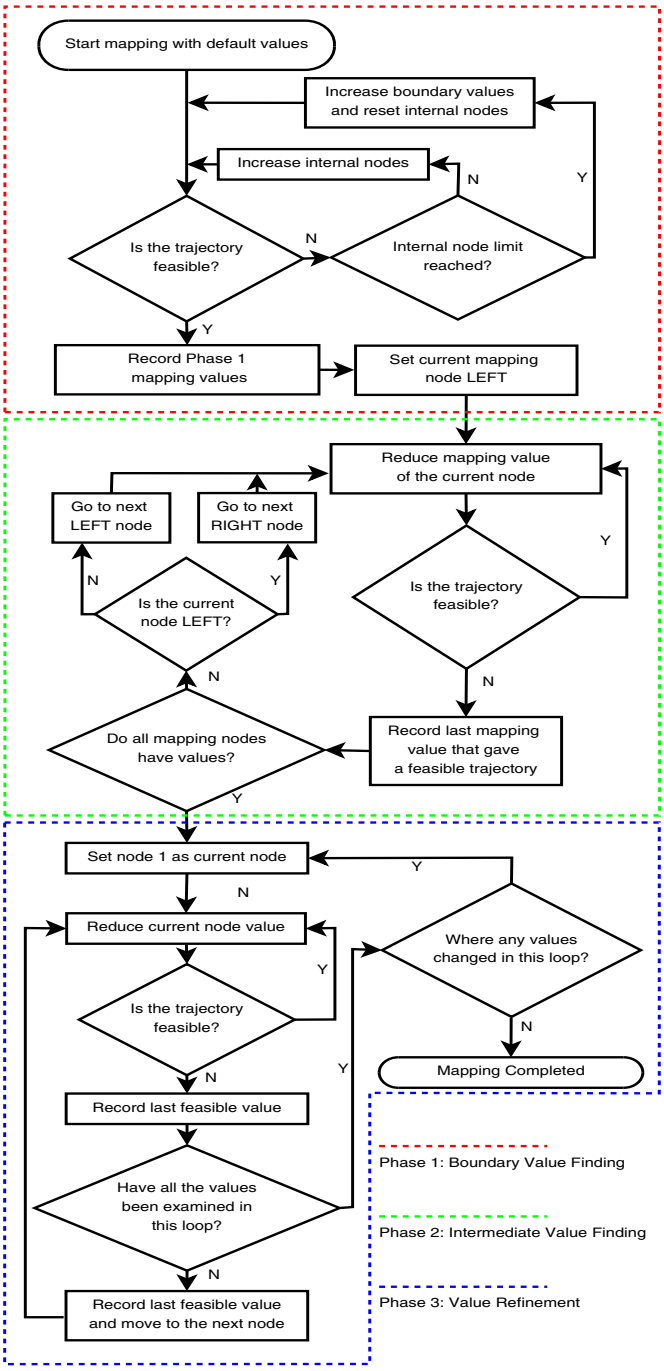

Fig. 1: Flowchart for the mapping algorithm that finds the mapping function.

IV. Results

Two cases are used to demonstrate the method. The first is a planar trajectory mission that was chosen for simplicity of demonstration. The second takes the output generated by an RRT algorithm to create a feasible 3D trajectory in a complex environment and demonstrate a more realistic application. This paper provides a summary of the full results which can be found at *http://personal.strath.ac.uk/jonathan.jamieson/IROS2016/*.

A. Planar Trajectory Mission

A planar trajectory is required to pass four waypoints: (0,0), (2,3), (10,3) & (25,6). The arbitrary boundary condition derivatives are:

$$\begin{aligned} &\dot{x}_1(0) = 3m/s, \;\; \ddot{x}_1(0) = 1m/s^2, \;\; \dot{x}_1(T) = 3m/s, \\ &\ddot{x}_1(T) = 1m/s^2, \;\; \dot{x}_2(0) = 1m/s, \;\; \ddot{x}_2(0) = -1m/s^2, \\ &\dot{x}_2(T) = 4m/s, \;\; \ddot{x}_2(T) = -2m/s^2, \end{aligned} \tag{16}$$

where T is the final time to be determined by the mapping function. The yaw angle was fixed at $\psi = 0$. The following kinodynamic limits were chosen:

$$\|\dot{x}\| < 7m/s, \;\; \|\ddot{x}\| < 5m/s^2 \tag{17}$$

Only the final trajectory is presented in this paper but the accompanying video shows all the iterations. The virtual trajectory position, speed and acceleration are shown in Fig. 3. The waypoints are covered but the acceleration and speed must be mapped to the time domain for a feasible trajectory. Twelve mapping nodes were used in the mapping function (Fig. 4b for this mission. The black markers denote places where mapping values were fixed. The nodes are more closely spaced at the edges because of LGL spacing. The relationship between τ in the virtual domain and time t is shown in Fig. 4a. The derivative of the mapping function $\dot{\lambda}$ against τ is plotted in Fig. 4c. The trajectory position, speed and acceleration in the time domain are plotted in Figures 5a, 5b and 5c respectively. The geometrical path is shown in Fig. 2, as previously mentioned this is identical in both the virtual and time domain.

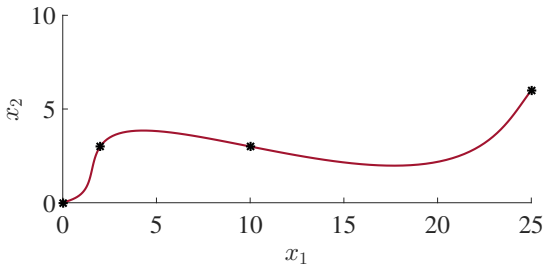

Fig. 2: The planar trajectory passing through 4 waypoints.

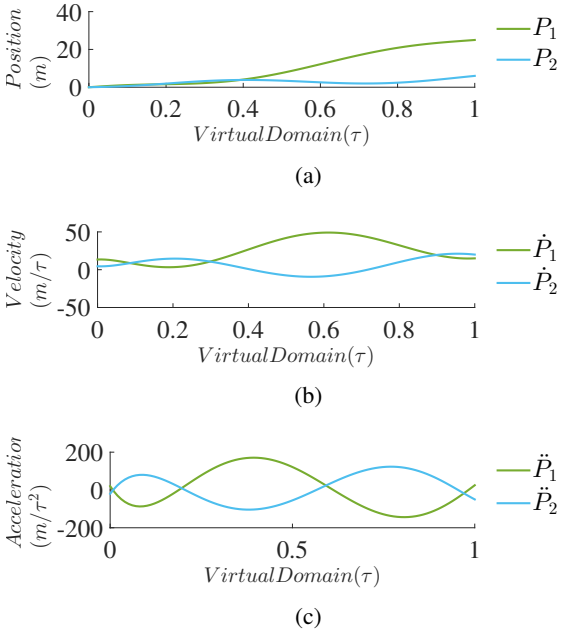

Fig. 3: (**a**) The position in the virtual domain (**b**) The velocity in the virtual domain (**c**) The acceleration in the virtual domain.

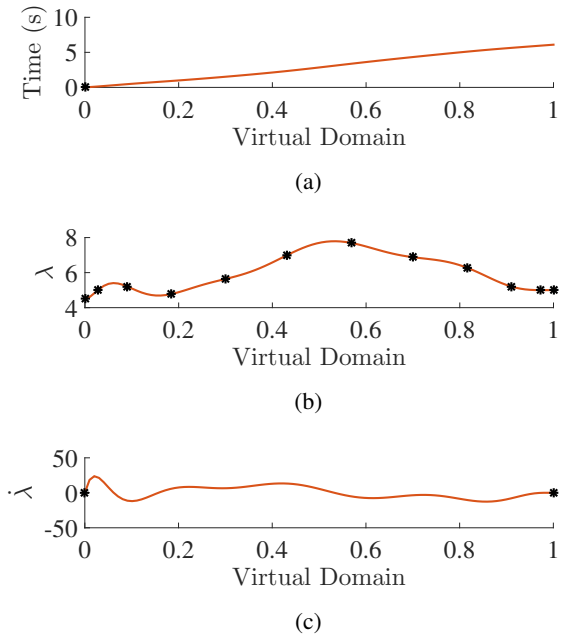

Fig. 4: (**a**) The mapped time against τ in the virtual domain. (**b**) The mapping function against τ. (**c**) The first derivative of the mapping function against τ.

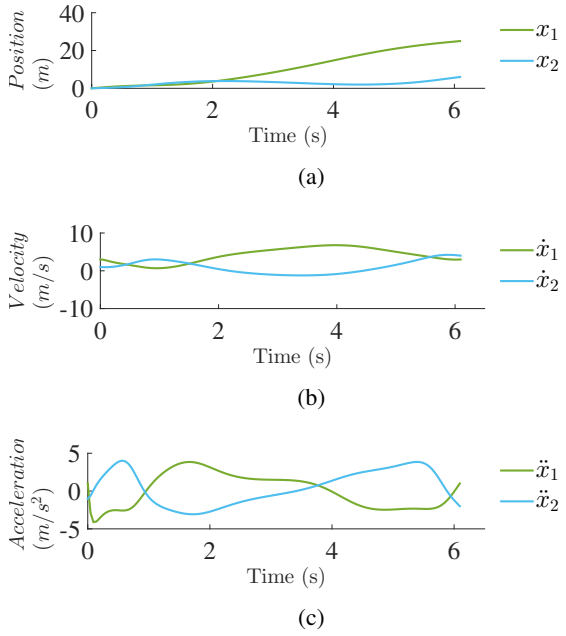

Fig. 5: (**a**) The position against time. (**b**) The velocity against time. (**c**) The acceleration against time.

B. RRT Trajectory

Fig. 6 shows a three-dimensional, rest-to-rest trajectory from $(0m,\ 1m,\ 0.5m)$ to $(10m,\ 2m,\ 0.5m)$ through an environment with three obstacles that are square walls with small rectangular cutouts. Additionally, the arena has volume $10 \times 3 \times 3m^3$ within which the quadrotor must remain at all times. The seven black markers mark the waypoints found using an RRT path planner after smoothing [17].

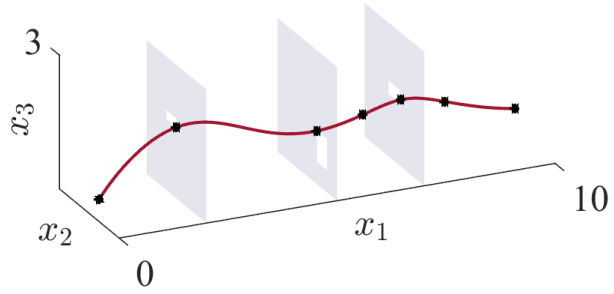

Fig. 6: Trajectory through an environment with obstacles planned using an RRT based method.

V. CONCLUSIONS

An algorithmic approach to finding near minimum-time trajectories for quadrotors is presented in this paper. We have focussed on trajectories that must cover multiple waypoints to enable flight in complex environments. Our method uses polynomials to describe the motion on the virtual domain before mapping into the time domain. The mapping algorithm allows for non-zero boundary derivatives and ensures the trajectories are within specified kinodynamic limits. We presented two examples, the first being a planar trajectory and the second a 3D problem with waypoints found using an RRT path planner.

Future work will look at optimising the mapping algorithm to find the mapping function faster and ensure it's suitable for onboard implementation. The dynamics of the quadrotor will be included using an inverse dynamics approach to improve the feasibility checking of the trajectory. Additionally the method will also be adapted for drone racing reference trajectories, a problem that requires a near minimum-time solution.

REFERENCES

[1] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 03, pp. 463–497, 2015.

[2] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proceedings of the International Symposium of Robotics Research (ISRR 2013)*, Singapore, 2013.

[3] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *arXiv preprint arXiv:1005.0416*, 2010.

[4] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, p. 0278364911434236, 2012.

[5] L. Babel, "Three-dimensional route planning for unmanned aerial vehicles in a risk environment," *Journal of Intelligent & Robotic Systems*, vol. 71, no. 2, pp. 255–269, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10846-012-9773-7

[6] J. E. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The international journal of robotics research*, vol. 4, no. 3, pp. 3–17, 1985.

[7] G. Boyarko, M. Romano, and O. Yakimenko, "Time-optimal reorientation of a spacecraft using an inverse dynamics optimization method," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 4, pp. 1197–1208, 2011.

[8] J. D. Biggs and L. Colley, "Geometric attitude motion planning for spacecraft with pointing and actuator constraints," *Journal of Guidance, Control, and Dynamics*, pp. 1–6, 2016.

[9] S. Dubowsky, N. Norri, and Z. Shille, "Time optimal trajectory planning for robotic manipulators with obstacle avoidance: a cad approach," in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, vol. 3. IEEE, 1986, pp. 1906–1912.

[10] I. D. Cowling, O. A. Yakimenko, J. F. Whidborne, and A. K. Cooke, "A prototype of an autonomous controller for a quadrotor uav," in *Control Conference (ECC), 2007 European*. IEEE, 2007, pp. 4001–4008.

[11] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," 2007.

[12] A. Caubet and J. Biggs, "A motion planning method for spacecraft attitude maneuvers using single polynomials," in *AAS/AIAA Astrodynamics Specialist Conference*, 2015.

[13] A. Caubet and J. D. Biggs, "An inverse dynamics approach to the guidance of spacecraft in close proximity of tumbling debris," in *International Astronautical Congress*, 2015.

[14] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 477–483.

[15] J. Pan, L. Zhang, and D. Manocha, "Collision-free and smooth trajectory computation in cluttered environments," *The International Journal of Robotics Research*, p. 0278364912453186, 2012.

[16] S. V. Parter, "On the legendre–gauss–lobatto points and weights," *Journal of scientific computing*, vol. 14, no. 4, pp. 347–355, 1999.

[17] M. Clifton, G. Paul, N. Kwok, D. Liu, and D.-L. Wang, "Evaluating performance of multiple rrts," in *Mechtronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME International Conference on*. IEEE, 2008, pp. 564–569.