Seminar SS 2018: Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control

Tridivraj Bhattacharyya (3035538)

Supervisor: Marcell Missura

July 9, 2018

Abstract

In this report, we analyze the work done by Liu et al. [3] for planning optimal motion while traversing an obstacle-cluttered environment. In earlier works on search-based planning even though multiple collision free geometric paths could be evaluated, this was done mostly with respect to time or space. The result was trajectories which might not be globally optimal or even feasible as they do not consider the system dynamics while planning. The paper addresses this problem by first generating motion primitives for the Quadrotor for fixed time intervals. In turn this allows creation of a graph or grid for all the reachable positions. Hence the problem becomes a Graph search one which is then solved by a path planning algorithm like A*.

1 Introduction

In order to understand the solution proposed in the paper, it is necessary to briefly discuss the following concepts:

1.1 Quadratic Programming

Quadratic Programming is a method to minimize objective functions which have additional linear constraints like equality or inequality. Let us consider the following function,

$$f(x) = \frac{1}{2}x^\intercal Q x + c^\intercal x$$

where, $c = n$ dimensional vector of real numbers
$Q = n \times n$ real symmetric matrix
$A = m \times n$ matrix of real numbers
$b = m$ dimensional vector of real numbers

The objective is to minimize $f(x)$.

$$\min f(x) \text{ such that } Ax \leq b$$

1.2 Step Function

A function comprised of a finite combination of several intervals is called a step function or a piece-wise constant function. Once plotted this resembles a stair case. A function $f : \mathbb{R} \to \mathbb{R}$ is a step function if,

$$f(x) = \sum_{i=0}^{n} \alpha_i \chi_{A_i}(x) \quad \forall x \in \mathbb{R} \tag{1}$$

where, $n \geq 0$
$\alpha_i \in \mathbb{R}$
A_i are the intervals
χ_A is the indicator function of A

$$\chi_A = 1_A = \begin{cases} 1 & \text{if} \quad x \in A \\ 0 & \text{otherwise} \end{cases}$$

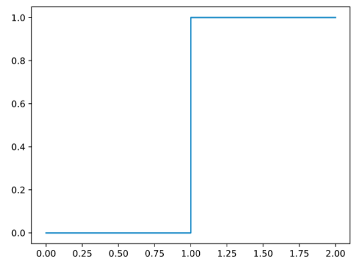


Figure 1: Step function

1.3 Occupancy Grid Mapping

Occupancy Grid mapping is a technique used in probabilistic robotics to plan a collision free path in an environment by navigating around any and all obstacles.
It provides a representation of the environment as an evenly spaced binary field of cells(2D) or cubes(3D). For example let us consider Figure 2. All cells which are occupied are represented as 1 or black while free cells are represented as 0 or white. Usually the goal is to calculate the probability of the map given the poses and measurement over a certain time interval.

$$\text{probability over map m} = p(m|z_{1:t}, x_{1:t}) \tag{2}$$

where $z_{1:t}$ = set of measurements over time t
$x_{1:t}$ = set of robot poses over time t

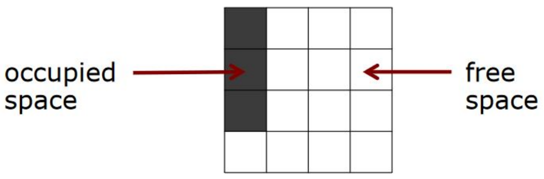


Figure 2: Occupancy Grid Mapping

1.4 A* algorithm

A* is one of the many path-finding algorithms which is used to find the shortest path between a source and goal in a grid or graph. It is pretty efficient in this regard by using a heuristic function to calculate the cost of moving between cells or nodes. The total cost of moving between nodes can be represented as,

$$f(n) = g(n) + h(n) \tag{3}$$

where, $g(n)$ = cost of moving from start node to current node
$h(n)$ = heuristic cost of moving from current node to goal node

The goal of A* is to find $f(n)$ of all possible nodes reachable from the current node. Then it selects the path greedily i.e. the one with the least cost. The trick is in selecting the heuristic. Depending on how the path can be expanded, Manhattan , Diagonal or Euclidean distance heuristics are used. There are also other ways to calculate the cost like the one used in the paper.

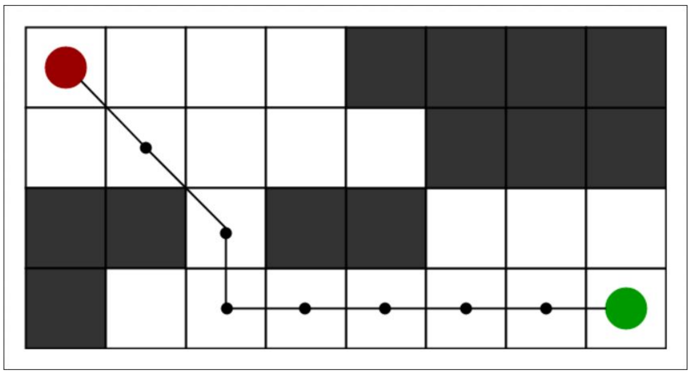


Figure 3: Path planned by A^*

1.5 Differential Flatness

Differential Flatness is a concept in systems theory which extends the controllability of linear systems to non-linear dynamic ones. Consider the following function for system states $x \in \mathbb{R}^n$ and control inputs $u \in \mathbb{R}^n$,

$$y = h(x, u, \dot{u}, \cdots, u^{(P)}) \tag{4}$$

where, $y \in \mathbb{R}^m$ = system output

The above system is considered differentially flat only when the the number of outputs is equal to the number of inputs and "all states and inputs can be determined from these outputs without integration". [2] The following conditions must hold for equation 4 to be a flat output,

$$x = x(y, \dot{y}, \ddot{y}, \cdots, y^{(q)})$$
$$u = u(y, \dot{y}, \ddot{y}, \cdots, y^{(q)})$$

1.6 Optimal Control

"Optimal control is the process of determining control and state trajectories for a dynamic system over a period of time to minimize a performance index." [4]

The goal is to optimize a cost function of the state and control variables for a dynamical system. This is very important in robotics to optimize motion planning. Now there are multiple cases of the optimal control problem. For our purpose we briefly discuss only the general case with a fixed time and no path constraints. Let us consider a system with state vector as $x(t)$ and control vector as $u(t)$. Then the optimal control problem is represented as,

$$J = \varphi(x(t_f)) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt \tag{5}$$

such that,

$$\dot{x} = f(x(t), u(t), t), \quad x(t_0) = x_0 \tag{6}$$

where, J = cost of the trajectory or motion also known as performance index,
t_f = final time,
t_0 = initial time,

φ = terminal cost function,
L = intermediate cost function
f = vector field

Now based on equations 5 and 6, another term that can be defined is the control Hamiltonian as follows,

$$H(x, \lambda, u, t) = \lambda^{\intercal}(t) f(x, u, t) + \int_{0}^{T} L(x, u, t) dt \tag{7}$$

Here, $t_0 = 0$ and $t_f = T$,
$\lambda(t)$ = state of co-state variables with same dimension as $x(t)$,

1.7 Pontryagins minimum principle

Based on the theory of optimal control, Pontryagin's minimum or maximum principle is used to determine the best way for a dynamical system to move from one state to another.This is particularly useful in solving minimum time problems.

Considering equations 5, 6 and 7, let $u(t) \in U$ which is the domain of all state vectors attainable i.e. permissible controls, the following constraint must be satisfied,

$$H(x^*(t), \lambda^*(t), u^*(t), t) \leq H(x^*(t), \lambda^*(t), u, t) \tag{8}$$

where, $x^*(t)$ = optimal state trajectory
$\lambda^*(t)$ = optimal co-state trajectory
$u^*(t)$ = optimal control

1.8 Controllability Gramian

This is a technique used in Control Theory to determine if a system is controllable or not. Given a set of linear equations,

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{9}$$

the Controballity Gramian is defined as,

$$W_c = \int_{0}^{\infty} e^{A_\tau} B B^{\intercal} e^{A_\tau^{\intercal}} d\tau \tag{10}$$

where, e_τ^A = state transition matrix in the time interval τ,

2 Summary

One of the biggest challenges in Robotics is planning movement. This includes path planning to reach certain goals along with motion as per its degrees of freedom. The problem itself is a non-trivial one. For differentially flat systems like Quadrotors, there are numerous techniques which have been used to plan paths in cluttered environments. However most of these work do not take into account the system dynamics in place. For example, to reach a certain point in space, a Quadrotor is bound by its capability (velocity, acceleration) to cover the distance withing a certain time. This includes assumptions that the goal is in free space and at least one path to it is unrestricted. Generally for such a problem, there are multiple solutions available. Despite that, all or even most of the solutions might not be feasible. It might be impossible to reach a certain way-point in a path or the trajectory planned by it could lead to sharp turns as jerk (momentum) comes into play.

2.1 Proposed Method

Liu et al. [3] proposed a solution for exactly this problem. In previous works, the problem is broken down to computing a collision free path and locally optimizing it to plan the movement. Here the problem is divided into two parts. First, conversion to a graph search problem by finding the reachable states for the Quadrotor. Then solving this graph search problem using an algorithm like A*. If one can consider motion primitives for the Quadrotor before planning the path, then it can lead to much more smoother trajectories. The solution proposed is as follows:

2.1.1 Creating Motion Primitives

The term motion primitives here mean the movement the Quadrotor can produce in any direction within a certain time. Before actually extracting them, the system needs to be defined with respect to the Linear Quadratic equations and its property of differential flatness. More details on this can be found in the paper by Murray et al. [2]

$$x(t) = [p_D(t)^\intercal, \dot{p}_D(t)^\intercal, \ddot{p}_D(t)^\intercal, \cdots, p_D^{(n-1)}(t)^\intercal] \tag{11}$$

where, $x(t) \in \chi =$ set of system states for the Quadrotor during time t,
$p_D^i(t) = i$-th derivative of $x(t)$

Now, χ represents the entire $\mathbb{R}^{3n}$ state space including both obstacle and free regions and the $n-1$-th derivatives. $x(t)$ is independent in each of the 3-D axes.

Therefore,
$P^{free} =$ free positions in state space,
$D^{free} =$ achievable system dynamics like velocity, acceleration etc.
$\chi^{free} \subset \chi = P^{free} \times D^{free} =$ free regions in state space,
$\chi^{obs} = \chi / \chi^{free}$ which means all χ which does not belong to χ^{free}

From equation 11, the first order polynomial can be defined as,

$$p_D(t) = \sum_{k=0}^{K} d_k \frac{t^k}{K!} = d_K \frac{t^K}{K!} + \cdots + d_1 t + d_0 \tag{12}$$

where, $D = [d_0, d_1, \cdots d_K] \in \mathbb{R}^{3\times(K+1)}$
Equation 11 is similar to the kinematic equation of $f(t+1) = f(t) + tv(t) + \frac{1}{2}t^2 a(t)$.

Equation (12) can be considered for a linear time invariant system with control input (e.g velocity provided to the Quadrotor) as $u(t) \in U = p_D^{(n)}(t)$ in 3-D space($\mathbb{R}^3$). Then the following linear equation becomes applicable for the system,

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{13}$$

where, $A = n \times n$ matrix,
B = n-dimensional vector

A smoothness effort for trajectories $J(D)$ is defined by taking into account all the control inputs,

$$J(D) = \int_0^T ||u(t)||^2 dt = \int_0^T ||p_D^{(n)}(t)||^2 dt$$

This denotes the cost or effort to move in a certain trajectory over total time T. The goal is to minimize this effort. However simple minimization might not work as the trajectory could be unrealistic. A weighted time is thus added to measure the smoothness against time.

$$\min_{D,T} J(D) + \rho T \tag{14}$$

such that equation 13 holds $\forall t \in [0, T]$
where, $x(0) = x_0$ = polynomials at initial state,
$x(T)$ = polynomials at goal state χ^{goal} after time T,
$x(t) \in \chi^{free}$ and $x(T) \in \chi^{goal}$
$\rho \geq 0$ = constant to weight total time T

In the paper, it is mentioned the solution for equation 14 is optimized using Pontryagin's minimum principle [section 1.7]. The optimal polynomial degree is found to be $K = 2n - 1$. This is assumed to be true for the following sections.

The control input $u(t)$ defined above is irrespective of any axes in physical space. Additionally the constraint that each of the polynomial trajectories must belong to free state space is what makes the problem so difficult to solve.

To create the motion primitives, the control set is first divided along each axis independently. Instead of planning a trajectory over the whole time T, it is sampled $\mu \in Z^+$ times for time intervals of $\tau > 0$. This limits the optimization problem from equation 14. What does it mean? The Quadrotor will move with a control input u_m for a time interval τ independently along each axis. As per the new discretization, $U_M := u_1, \cdots u_M \subset U$ are the motion primitives along each 3-D axis.

From equations 11 and 12,

$$u(t) = p_D^{(n)}(t) = \sum_{k=0}^{K-n} d_{k+n} \frac{t^k}{k!} \equiv u_m \quad \forall t \in [0, \tau]$$

For an initial condition of $x_o := [p_0^\intercal, v_0^\intercal, a_0^\intercal \cdots]^\intercal$, integrating over control input u_m, equation 12 becomes equal to,

$$p_D(t) = u_m \frac{t^n}{n!} + \cdots + a_0 \frac{t^2}{2} + v_0 t + p_0$$

Using the linear equation 13, the resulting trajectory can be defined as,

$$\dot{x}(t) = e^{At} x_0 + \left[\int_0^t e^{A(t-\sigma)} B d\sigma\right] u_m = F(t) x_0 + G(t) u_m$$

The cost function defined previously also changes to,

$$C(s) = J(D) + \rho\tau = (||u_m||^2 + \rho)\tau \tag{15}$$

where, s = current state of the system $\in \chi$. The time interval τ and motion primitive u_m are fixed throughout.

2.1.2 Conversion to Graph Search

The motion primitives and discretization in state space χ tells exactly which positions the Quadrotor can reach. A graph is created out of this information. If $R(s)$ be the set of reachable states lying in χ^{free} from s and $C(s)$ be their cost, trajectories to $R(s)$ can be calculated. The graph is expanded for all motion primitives at each time interval τ till the goal state is reached. The detailed algorithm can be found in the original paper[section III B] [3]. Since the trajectories are computed after optimizing equation 14, these are also considered to be optimal.

To formulate the problem as a graph search problem, the total time is broken down to $N \in Z^+$ intervals. Therefore the total time taken is $T = N\tau$. The control inputs $u_k \in U_m$ is constant over each time interval. Thus this is a piece-wise constant or step function [Section 1.2]. In terms of step function, the control input can be represented as,

$$u(t) = \sum_{k=0}^{N-1} u_k 1_A$$

where, $A \Rightarrow t \in \{k\tau, (k+1)\tau\}$

Combined with the independent motion primitives, one can get all trajectories from current state s to a goal. What remains would be to estimate the shortest path from all of them. This is where the A* algorithm comes in.

2.1.3 Designing the Heuristic

To apply A* algorithm one needs a heuristic to compute the cost. A straightforward euclidean heuristic is not applicable to the problem here as there are way too many system constraints to simply follow a straight line till the goal.

Using the reformulated time constraints and motion primitives, equation 15 is further changed to,

$$\min_{N, u_{0:N-1}} \left(\sum_{k=0}^{N-1} ||u_k||^2 + \rho N \right) \tau \tag{16}$$

such that, $x_k(t) = F(t)s_k + G(t)u_k \subset \chi^{free} \quad t \in [0, \tau]$
$x_k(t) \subset \chi^{free} \quad \forall k \in 0, \cdots, N-1$
$s_{k+1} = x_k(t) \subset \chi^{free} \quad \forall k \in 0, \cdots, N-1$
$s_0 = x_0, \, s_N = \chi^{goal}$
$u_k \in U_m \quad \forall k \in 0, \cdots, N-1$

To simplify the problem, equation 16 is solved in only lower dimension namely velocity and acceleration profile. For the Quadrotor to reach a point within the time interval τ, it is restricted by its max velocity or acceleration.

If only velocity is considered, the time taken to reach a state p_f from an initial state p_0 can be termed as, $T_v := \frac{||p_f - p_0||_\infty}{v_{max}}$

For acceleration controlled input, the time is formulated as T_a which is bound by a_{max}. The goal would be to find,

$$\min_{T_a, a(t)} T_a$$

such that, $||a(t)|| \leq a_{max} \quad \forall t \in [0, \tau]$
$p(0) = p_0 \quad v(0) = v_0$
$p(T_a) = p_f \quad v(T_a) = v_f$

The time profile T_a can be further divided individually as per each 3-D axis $[T_a^x, T_a^y, T_a^z]$. The total time taken to traverse the shortest path from current state to goal would then have to be greater than these minimum time profiles. Since it is very complicated to implement this in higher order derivatives like jerk, only velocity and acceleration is considered. Therefore equation 14 is relaxed by dropping constrains $x(t) \in \chi^{free}$ and $u(t) \in U$,

$$\min_{D,T} J(D) + \rho T \tag{17}$$

such that, $\dot{x}(t) = Ax(t) + Bu(t) \quad \forall t \in [0, T]$
$x(0) = x_0 \quad x(T) \in \chi^{goal}$
$T \geq T_v$ for velocity profile

The cost $C^*(x_0, \chi^{goal})$ is therefore greater than ρT_v. This provides the velocity profile heuristic as,

$$h_1(s_0) = \rho T_v = \frac{\rho ||p_f - p_0||_\infty}{v_{max}}$$

The above heuristic while being simple might not be enough for planning smooth trajectories. The reason is it does not take into account the system constraints. A second heuristic is computed by applying the solution for Linear Quadratic Minimum Time Problem on equation 17. Again the derivation for this is not shown here and is assumed to be true. By defining a Controllability Gramian [Section 1.8] and a function $\delta_T := x_f - e^{AT}x_0$ the minimum time is obtained either from the previous heuristic or by solving the following equation,

$$-\frac{d}{dT}\{\delta_T^\intercal W_T^{-1} \delta_T\} = 2x_f^\intercal A^\intercal W_T^{-1} \delta_T + \delta_T^\intercal W_T^{-1} B B^\intercal W_T^{-1} \delta_T = \rho \tag{18}$$

where, W_T is the controllability gramian $= \int_0^T e^{At} B B^\intercal e^{A^\intercal t} dt$

From previous sections, ρ is the constant used to measure smoothness against time. The optimal control and heuristic cost using equation 18 are defined as below,

$$u^*(t) := B^T e^{A^\intercal (T-t)} W_T^{-1} \delta_T \tag{19}$$

$$h_2(x_0) = \delta_T^\intercal W_T^{-1} \delta_T + \rho T \tag{20}$$

The polynomial coefficients $D \in \mathbb{R}^{3 \times (2n)}$ till $n-1$ is equal to the initial trajectory and from n to $(2n-1)$ can be obtained with the Hamiltonian.

$$d_{0:n-1} = x_0 \text{ and } d_{n:2n-1} = \delta_T^\intercal W_T^{-1} e^{AT} H^\intercal$$

where $H \in \mathbb{R}^{(3n)\times(3n)}$ is the control Hamiltonian with, $H_{ij} = \begin{cases} (-1)^j, & i = j \\ 0, & i \neq j \end{cases}$

From section 2.1.1, the state space χ belongs to $\mathbb{R}^{3n}$ dimensions. Differing the value of n provides the control inputs in higher state space. eg. For $n = 1$, the controls are obtained in velocity space and for $n = 2$, in acceleration controlled space. The optimal cost for each of these controls are,

1. $C^* = \frac{1}{T}||p_f - p_0||^2 + \rho T$ where $\chi \subset \mathbb{R}^3$
2. $C^* = \frac{12||p_f - p_0||^2}{T^3} - \frac{12(v_0+v_f)(p_f-p_0)}{T^2} + \frac{4||v_0||^2 + v_0 v_1 + ||v_1||^2}{T} + \rho T$ where $\chi \subset \mathbb{R}^6$

The optimal cost under acceleration profile provides the minimum time $T^* = C^*(T^*)$ Detailed derivation of the above is available in section III-D of the paper. [3]

2.1.4 Planning and Refining the Trajectory

For each of the trajectories planned above, it is necessary to check if the entire geometric shape of the trajectory lies in free state space P^{free} and is not limited by the system constraints D^{free}. Breaking down the trajectory into checkpoints and validating if each of these sections lie in free space tells if it is collision free or not. An occupancy grid map is constructed from this which along with the previous graph search solution provides feasible shortest paths to the goal.

There is still a further problem that the Quadrotor may not exactly follow this planned trajectory cause of unexpected system dynamics like wind shear. The problem is solved by the solution provided in the work by Mellinger et al. [1]. Additionally the problem is considered an unconstrained Quadratic Programming one [Section 1.1]. In this case, the following equation needs to be solved,

$$\min_{D} \sum_{k=0}^{N-1} \int_0^{\tau_k} ||p_{D_k}^{(n)}(t)||^2 dt \tag{21}$$

such that, $x_0(0) = s_0 =$ starting state
$x_{N-1}(\tau_{N-1}) = s_g =$ goal state
$x_{k+1}(0) = x_k(\tau_k) \quad k \in 0, \cdots, N-2$
$p_{D_k}(\tau_k) = p_k \quad k \in 0, \cdots, N-1$
where, $p_k, k \in 0, \cdots, N-1$ are the intermediate checkpoints. Generating trajectories in higher dimensions are also possible using this method. The authors have however warned in the paper that despite this refinement step, the trajectory can still be dangerous. The exact reason is not mentioned and is estimated that some sort of collision is still possible along the trajectory or that calculations are much more complicated in higher dimensions.

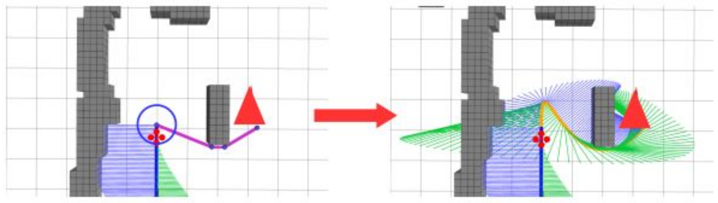
The last step in the planning process is re-planning. Environments in the practical world are usually way too large for any sensor to map in one sweep. Therefore as more information is gathered, the path needs to be re-planned to navigate around obstacles which were not detectable previously. Receding Horizon Control (also known as Model Predictive Control) is used for this step which is a technique to run the optimization as defined previously in an iterative manner. Therefore anytime any new obstacles pop up, it will be possible for the Quadrotor to avoid it.

2.2 Experimental Results

The proposed solutions was tested in both 2-D and 3-D environments with both heuristics as defined in Section 2.3. It was found that for systems with a low maximum velocity, the velocity control works pretty well. However with acceleration control, computation was faster and provided more optimal trajectories. The system was tested with both heuristics using A* algorithm and for reference with Dijkstra by setting the heuristic to 0. By definition itself, A* is primarily an improvement over Dijkstra.

Furthermore, the system was tested in jerk-controlled space which was found to be much slower than acceleration. One would need to calculate the roots for the polynomials in the cost function which gets harder as the value of n increases. In higher dimensions the heuristic becomes very hard and expensive to compute.

The two primary testing environments mentioned in the paper are a cluttered corridor and a goal area behind a wall. In the second case, the Quadrotor had to circle around the wall to reach the goal. This is in contrast to other works where there was a sudden deceleration and then turn to reach the goal. In both cases the results are admirable as the Quadrotor navigates very efficiently in a smooth manner using the re-planning steps to avoid all the obstacles.


(a) Old Path-based approach

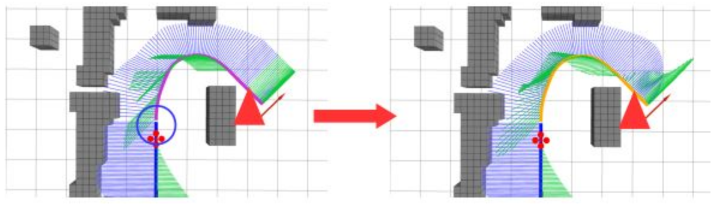

(b) New Method

Figure 4: Trajectories for Experiment 1

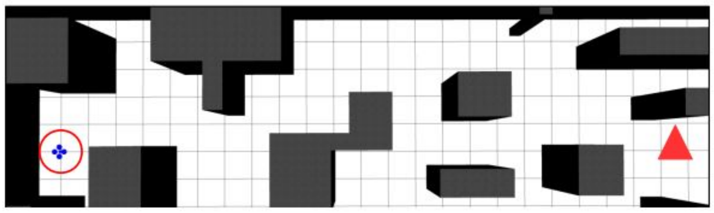
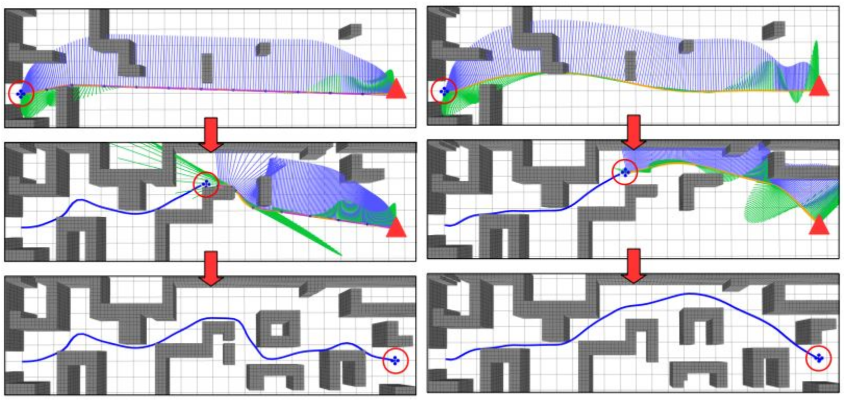


Figure 5: Environment for Experiment 2



(a) Old Path-based approach (b) New Method

Figure 6: Trajectories for Experiment 2

2.3 Assessment

The paper has summarized the work as energy saving and reasonably optimal. Its claim looks valid based on the experiments as the planning seems pretty instantaneous. The computation time is reasonably fast considering that at the end it is primarily an optimization problem. The framework is also claimed to be reusable with other path planning algorithms and also with any differentially flat systems.

3 Discussion

As mentioned earlier, path-planning in the physical world in robotics is a highly non-trivial problem. Most previous research in this area has taken an approach to solve the problem by formulating it only with respect to time and geometric space. In the described work, they have gone one step further by considering the dynamic constraints of the system itself. This makes sense in practical terms as it is of utmost importance for any system to work within its limitations or make the entire process more expensive. eg. A sudden change in direction could cause extra steps of re-planning thereby increase space and time required for computations. Breaking down the problem into an optimization and graph search one independently makes it more clinical. It eliminates considering trajectories where sudden jerks may actually occur. It not only provides an optimal path to a goal but notifies the system on the best way to approach it.

Mathematically, the initial derivations are well described. By defining the initial problem along a fixed total time and then gradually breaking it down by intervals to create checkpoints is relatively convenient to follow. The derivations for the optimization of equation 14 is unclear in the absence of open source or free material but it is assumed to be true. The final sections of the solution where concepts from optimal control are applied needs some in-depth research to understand especially if one is not familiar with Control Theory. Having mentioned that, reproducing the results seem possible without a deep understanding of the derivations as long as one can figure out the calculations for the heuristics. The generic nature of the work makes it very convenient. The authors have mentioned that it can be used to apply other path planning algorithms along with trajectory generation for all differentially flat systems.

This would make it highly valuable for future work in robotics w.r.t dangerous or unstable environments (eg. robots deployed for surveillance in urban areas hit with natural disasters).

The experimental results prove the claim of the paper to be quite accurate. However there are some scenarios which would be helpful in understanding how efficient the solution is. Urban areas are not only cluttered but can also contain dynamic obstacles like humans or animals moving around. It would be interesting to see if the algorithm performs as efficiently and fast as claimed in such scenarios. It is also not clear how the time interval τ is defined. Whether this is based on selective environments and is a matter of trial of error has not been discussed in the paper. Intuitively if the value is too low it could lead to higher number of iterations and therefore more computation time. If the value is too high, it could lead to unnecessarily larger geometric paths. eg. By having a larger time interval, one of the checkpoints along a smooth trajectory could be lying along an obstacle. While a smaller interval could plan a feasible enough path around this obstacle, a larger interval would probably mean going too much out of the way to avoid it. It is also not clear whether this interval evolves over time and adapts to the environment. The first glance at the code available on github [https://github.com/sikang/motion_primitive_library.] does not reveal any changes during the runtime. This could be a possible improvement or a matter of future work.

4 Conclusion

The paper proposes a method for finding optimal paths by a Quadrotor taking system dynamics into account. It fuses together concepts of Quadratic Programming, Optimal Control and Graph Search to provide a neat and efficient solution for an ongoing area of research in Robotics. Firstly it takes the problem and defines motion primitives which is later broken down to independent components along each axis. This results in a graph for which the authors implement A* after defining their own heuristic for shortest path. While the mentioned experiments show the method to be working well, further testing and clarifications are required to determine the overall efficiency in the real world. However the generic nature of the provided framework should make it possible for testing by others in various practical scenarios.

5 References

References

[1] Daniel Mellinger et al. “Minimum snap trajectory generation and control for quadrotors”. In: *2011 IEEE International Conference on Robotics and Automation* (2011). DOI: `https://doi.org/10.1109/ICRA.2011.5980409`.

[2] Richard M. Murray et al. “Differential Flatness of Mechanical Control Systems: A Catalog of Prototype Systems”. In: *1995 ASME International Mechanical Engineering Congress and Expo* (1995). DOI: `http://www.cds.caltech.edu/~murray/preprints/mrs95-imece.pdf`.

[3] Sikang Liu et al. "Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017). DOI: `https://doi.org/10.1109/IROS.2017.8206119`.

[4] V. M. Becerra. "Optimal control". In: *Scholarpedia* 3.1 (2008). revision #124632, p. 5354. DOI: `10.4249/scholarpedia.5354`.