Two-Phase Scalable Mixed-Integer Path Planning for UAVs

Jorik De Waen

April 21, 2017

Contents

1 Introduction

As a consequence of ever-increasing automation in our daily lives, more and more machines have to interact with and unpredictable environment and other actors within that environment. One of the sectors that seems like it will change dramatically in the near future is the transportation industry. Autonomous cars are actually starting to appear on public roads, autonomous truck convoys are being tested and large retail distributors like Amazon are investing heavily into delivering order by drones instead of courier. While these developments look promising, there are still many challenges that prevent these systems from being widely deployed.

One such challenge, which is especially important for areal vehicles, is path planning. Even though most modern quadrocopters are capable of flying by themselves, they are unable to generate a flight path that will get them to their destination reliably. Classic graph-based shortest path algorithms like Dijkstra's algorithm its many variants fail to take momentum and other factors into account. Mixed Integer Linear Programming (MILP) is one approach that shows promising results, however it is currently severely limited by computational complexity.

1.1 Motivation and Goal for the Thesis

One of the main advantages of using a constraint optimization approach like MILP is that they are extremely extendable by design. A system based on this can be deployed in many different scenarios with different goals and constraints without the need for significant changes to the algorithms that drive it. The solvers that construct the final path are general solvers which take constraints and a target function as input. This input can be generated by end users in the field to match their specific requirements, making the software controlling the drones as flexible as the hardware.

That flexibility is also the main limitation of using constraint optimization. The solvers are general purpose, which make them very slow compared to more direct approaches. They need to be carefully guided solve all but the most basic scenarios in a reasonable amount of time. While there have been some good results on small scales, I could not find any attempts at planning paths on the order of kilometers or more. Practical use cases involving drones often involve several minutes of flight and can cover several kilometers, so a path planner must be able to work at such a scale. This is the main goal of this thesis: To demonstrate how a MILP approach can be scaled to scenarios with a much larger scope, while preserving the advantages that make it interesting.

1.2 Structure of the Thesis

Section 1.3 summarizes the previous work that has been done in the field. The previous work in the field shows a common design to modeling the path planning problem as a MILP problem. This design forms the core of the approach in this thesis as well. Section **??** shows the implementation of this common design and explores the critical limitations to this approach.

Section 3 proposes a solution to these limitation. By finding a rough initial path, the planning problem can be split into smaller segments. Solving these segments on their own is significantly easier and can still enforce all the constraints. This approach is much faster than previous techniques, but at the cost of no longer finding the global optimum.

During the development of this algorithm, finding and solving bugs and other unwanted behavior proved to be a significant challenge. A visualization tool was developed to make it easier to see how the algorithm operates. Section 6 goes into detail of how nearly every variable in the MILP problem

was visualized and how this information can be interpreted.

To demonstrate the flexibility of the approach, section **??** showcases some possible extensions that can be added with relative ease. Some of these have been fully implemented to look at the impact on the solution. This section should demonstrate that the path planner discussed in this thesis is a modular strategy built out of several different algorithms. The specific algorithms discussed are just one way of doing things, and can be easily swapped out for other, more advanced, algorithms.

Section **??** analyzes the performance of the path planner in several different scenarios. It also looks at how the extensions which have been implemented affect the both the performance and quality of the planner. Finally, section **??** summarizes the main observations in this thesis and concludes whether or not the goals have been realized.

1.3 Context

Mixed integer linear programming is and extension of linear programming. In a linear programming problem, there is a single (linear) target function which needs to be minimized or maximized by the solver. A problem typically also contains a number linear inequalities which constrain the values of the variables in the target function. When all the variables are real, problems like

$$\begin{aligned}&\text{maximize} \quad 100x_1 + 125x_2 \\ &\text{subject to} \\ &3x_1 + 6x_2 \leq 30 \\ &8x_1 + 4x_2 \leq 44\end{aligned}$$

Figure 1: An example linear problem

this can be solved in polynomial time. However, only convex search spaces can be modeled this way. By adding obstacles to the world the drone has to navigate, the search space becomes non-convex. To express non-convex search spaces, integer variables are required[8]. A linear problem with integer variables is called a mixed integer linear problem. These integers make it possible to model logical expressions, which in turn enable approximations of non-linear function to be used. Another consideration is how to model time. Constraint solvers typically have no concept of time, so this needs to be modeled as well.
The most common way to do this is the way Schouwenaars et al[8] originally

modeled time: A limited amount of discrete timesteps. Planning using MILP is very computationally expensive, so an attempt by Bellingham[1] was to use a receding horizon and only plan a small part of the path at a time.
Flores[5] and Deits et al[3] do not use discretized time, but model continuous curves instead. This no longer uses linear programming, but quadratic, conic or higher.
Several different papers[4, 6, 2, 7] show how the output from algorithms like these can be translated to control input for an actual physical vehicle. This demonstrates that, when they properly model a vehicle, these path planners need minimal post-processing to control a vehicle.
TODO:PRM and RERT,

2 Modeling Path Planning as a MILP problem

2.1 Introduction

TODO: REWRITE MILP is a form of mathematical programming, form of declarative programming. contrast with imperative programming: say what the solution looks like instead of how to get there. Mathematical programming: subset of declarative programming where problem is defined as mathematical problem. Solver computes result.
Big advantages:
- dont need to say how it is done, focus is on precisely modeling problem.
- flexible: can easily change rules: make problem more restrictive or relaxed, change the goal, etc "just works"
- can be really fast thanks to work on solvers

disadvatages:
- no free lunch (find reference): for anything more than very basic cases, solver cannot guarantee that a solution will be found any faster than random search
–¿ cant rely on just the solver doing the work, problem needs to be stated in a way that guides solver in the right direction –¿ even when careful, as problem becomes more complex, solvers struggle
- hard to understand: solver finds solution or not. –¿Solution is optimal, but if it is not as expected it does not give any information why. –¿when

no solution: can show which constraint failed, but problem is not necessarily there. complex interplay between constraints is extremely hard to debug

2.2 Terminology

TODO: FILL OUT variable constraint solution space feasible region solver (cplex) convex

2.3 Importance of integers and convexity

TODO: REWRITE linear programming one of the most simple forms of mathematical programming. Only linear functions allowed, all variables have a real domain. If feasible region exists: inherently convex. When a problem is convex, typically efficient solvers can be written (TODO: find source). This is the case for LP. LP problems can be solved reliably and quickly.
Problem: only real variables is very limiting. Logical operators are impossible to model. Result is that not every problem can be modeled without integers. Adding integers seems like a minor change, but it no longer guarantees that the feasible region is convex. When the problem is no longer convex, it becomes intractible. An intractible problem is a problem for which we do not have a strategy that will on average find the solution faster than random guessing.
Path planning when obstacles are present inherently needs integer variables and thus is an intractible problem. The goal of this thesis is to make this approach to path planning scalable. This will rely on two concepts:// - The solvers are on average not faster than random, but that's the average of all problems. We are only concerned with a very specific kind of problem. By being aware of how the solver works and simply experimenting with different representations, the problem can be solved much faster
- We can cheat. The problem can be greatly simplified and approximated. This means that the solution is no longer guaranteed to be optimal, but as long as it is good enough this is not a problem.

2.4 Basic Path Planning MILP Model

The path planning problem can be represented with discrete timesteps with a set of state variables for each epoch. The amount of timesteps determines the maximum amount of time the vehicle has in solution space to reach its goal. The actual movement of the vehicle is modeled by calculating the accelleration, velocity and position at each timestep based on the throttle (in each axis) and state variables from the previous time step.

$time_0 = 0$
$time_{t+1} = time_t + \Delta t, \quad 0 \leq t < N$

$\boldsymbol{pos}_0 = \boldsymbol{pos}_{start}$
$\boldsymbol{pos}_{t+1} = \boldsymbol{pos}_t + \Delta t * \boldsymbol{vel}_t \quad 0 \leq t < N$

$\boldsymbol{vel}_0 = \boldsymbol{vel}_{start}$
$\boldsymbol{vel}_{t+1} = \boldsymbol{vel}_t + \Delta t * \boldsymbol{acc}_t \quad 0 \leq t < N$

$\boldsymbol{acc}_t = \boldsymbol{throttle}_t * \boldsymbol{acc}_{max} \quad 0 \leq t \leq N$

The problem also needs a goal function to optimize. In this model, the goal is to minimize the time before a goal position is reached. Optionally, there is also a goal velocity that needs to be matched when the vehicle reaches the goal. Reaching the goal constraints causes a state transition from not being finished to being finished. Modeling state transitions directly can be error-prone, so Lamport's[**?**] state transition axiom method was used. In this simple model it is still possible to model the state transition directly, but the goal of the thesis is to provide a flexible and extensible approach.

The most challenging part of the problem is modeling obstacles. Any obstacle between the vehicle and its goal will inherently make the search space non-convex. Because of this, integer variables are needed to model obstacles. The most common way to do this is to use the "Big M" method to model a polygon. The size of the vehicle needs to be taken into account. Assuming the polygon is convex and the vertices of the polygon are listed in counter-clockwise order, the following constraints model an obstacle:

$$minimize \quad N - \sum_{t=0}^{t \leq N} fin_t$$
$$fin_0 = 1$$
$$fin_{t+1} = fin_t \vee cfin_{t+1}, \quad 0 \leq t < N$$

$$cfin_{pos,t} = \bigwedge_{i=0}^{i<Dim(\boldsymbol{pos}_t)} |pos_{t,i} - pos_{goal,i}| < pos_{tol}, \quad 0 \leq t \leq N$$

$$cfin_t = cfin_{pos,t} \quad 0 \leq t \leq N$$

OBSTACLE CONSTRAINTS

2.5 Characteristics of the basic model

Nothing new in basic model, motivate changes based on flaws in the model. demonstration with simple scenario (benchmark and spiral) where basic model already struggles. Approximate amount of integer variables needed, use as representation of nonconvexity of problem.
Also show issue with time allowed: Need to allocate way more time than is needed, making the problem even harder to solve. Limiting time makes problem faster to solve, but cannot know without prior information.
Demonstrate issue with corners. Corners make execution time go up even as amount of integer variables stays the same. If no corners are needed, feasible space stays more convex even though integer values are same. Nuance that integer variables are only apprixmation of non-convexity, convexity is real property that matters.
Conclude: need to reduce amount of obstacles, need to reduce timesteps, limit amount of corners at a time (so problem becomes more convex). Dividing problem into smaller pieces tackles all these goals.

3 Segmentation of the MILP problem

3.1 Introduction

TODO:REWRITE Key to making MILP scale is dividing the problem into smaller pieces. Goal cannot be reached in a single segment, so needs to be guided some other way. Need to consider that multiple corners in single segment is problematic, so segments should minimize the amount of corners. Will use theta* and detect corners, use corners to construct segments.
Smaller segments allow lower upper bound on time needed, obstacles still an issue. Maximum possible distance vehicle can travel is first approximation, but still not enough in dense cities. Need to significantly cut down on amount of obstacles safely. Will demonstrate that not every obstacle is equally important, how to find the important obstacles and safely compress other obstacles using genetic algorithm.
Stress that the solution chosen can be swapped out. What's written here is one option.

3.2 Guiding the segmented path

The first issue is that because the goal cannot be reached immediately, the goal function needs to change.
One option is to simply get as close as possible to the goal during each segment. Because the distance that can be traveled during a segment is limited, the amount of obstacles that need to be modeled is also limited. This works well when the world is very open with little obstacles. However, this greedy approach is prone to getting stuck in dead ends in more dense worlds like cities.
The second option is finding a complete path using a method that's easier to compute. An algorithm like A* can be used to find a rough estimation for the path. This A* path is the shortest path, but does not take constraints or the characteristics of the vehicle into account. A very curvy direct path may be the shortest, but a detour which is mostly straight and allows for higher speeds may actually be faster.
It is possible to use more advanced algorithms that model more of the constraints. This will significantly improve the speed and quality of the constraint optimization step, but it will also come at a performance cost. A balance needs to be found between the preprocessing step and the constraint

optimization step. The preprocessing step needs to do just enough so the optimization step can be solved in an acceptable amount of time.

3.2.1 Thetha* implementation

I have decided to use Theta*. This is a variant of A* that allows for paths at arbitrary angles instead of multiples of 45 degrees. The main reason for this is that it eliminates the "zig zags" that A* produces. This makes the next step of the preprocessing much easier.

3.2.2 Scalability of Theta*

TODO: REWRITE Theta* provides the information that makes it possible to make MILP fast, but Theta* in itself is intractable. Why is this not simply moving the issue?
Theta* is indeed intractable, but still much better than MILP. Holomomic vs non-holonomic. Concept of navigation mesh as part of map, often used in video games to allow AI to navigate large worlds. Also same concept behind PRM. Only needs to pre precomputed once per map. Can serve as heuristic for Theta*, or completely replace it.

3.3 Detecting corner events

With an initial path generated, the next problem is dividing it into segments. Dividing the path into equal parts presents problems, because when solving each segment, the solver has no knowledge of what will happen in the next segment. This is especially problematic when the vehicle needs to make a tight corner. If the last segment ends right before the corner, it may not be possible to avoid a collision.
In Euclidian geometry, the shortest path between two points is always a straight line. When polygonal obstacles are introduced between those points, the shortest path will be composed of straight lines with turns at one or more vertices of the obstacles. The obstacle that causes the turn will always be on the inside of the corner. This shows why corners are important for another reason: They make the search space non-convex. For obstacles on the outside of the corner it is possible to constrain the search space so it is still convex,

but this is not possible for obstacles on the inside.
Because of these reasons, isolating the corners from the rest of the path is advantageous. With enough buffer before the corner, the vehicle is much more likely to be able to navigate the corner successfully. It also means that the computationally expensive parts of the path are as small as possible while still containing enough information for fast navigation through the corner.
The reason for using Theta* becomes clear now. Every single node in the path generated by the algorithm is guaranteed to be either the start, goal or near a corner. A corner can have more than one node, so nodes which turn in the same direction and are close to each other are merged into a single corner event. A corner event is also generated for nodes which are alone.

3.3.1 Tight Coupling with Theta*

This algorithm relies on assumptions from Theta*, so if algorithm changes, this will need to be updated.

3.4 Generating path segments

These corner events are in turn grown outwards to cover the approach and departure from the corner. How much depends on the maximum acceleration of the vehicle. As a rule of thumb: If the vehicle can come to a complete stop from its maximum speed before the corner, it can also successfully navigate that corner. When corners appear in quick succession, their expanded regions may overlap. In that case, the middle between those corners is chosen. Long, straight sections are also divided into smaller path segments.
One of the main goals of segmenting the path is to reduce the amount of obstacles. This means that every segment has a set of obstacles associated with it, being the obstacles that need to be modeled in the optimization step. Not only the obstacle that "causes" the corner is important, but obstacles which are nearby are important as well. Obstacles on the outside of the corner also may play a role in how the vehicle approaches the corner. To find all potentially relevant obstacles, the convex hull of the (Theta*) path segment is calculated and scaled up slightly. Every obstacle which overlaps with this shape is considered an active obstacle for that path segment. The convex hull step ensures that all obstacles on the inside of the corner are included, while scaling it up will cover any restricting obstacle on the outside of the corner.

3.5 Generating the active region for each segment

Even though the most important obstacles are taken care of, all other obstacles also need to be represented. To do this, a convex polygon is grown around the path. This polygon may intersect with the active obstacles (since they will be represented separately), but may not intersect any other obstacle. The polygon is grown using a genetic algorithm. It uses a single mutator which nudges the vertices of the polygon while ensuring it stays convex and does not intersect itself or any non-active obstacle. The genetic algorithm is just one way to generate the convex polygon which represents the active region. Deits and Tedrake[3] have demonstrated how another algorithm can solve the same problem.

4 Extensions

4.1 Abs speed

4.2 Max speed

4.3 Min speed

4.4 Indicator variables

4.5 Max time

4.6 Backtracking

5 Improving the MILP problem

Vehicles have a maximum velocity they can achieve. Calculating the velocity of the vehicle means applying Pythagoras' theorem on the axis of the velocity vector. This is not possible using only linear equations. However, it can be approximated to an arbitrary degree using multiple linear constraints. The components of the velocity vector can be positive or negative, but only the absolute value matters for the actual velocity.

ABS
MAXSPEED

Because obstacles make the problem non-convex and thus require integer constraints to model, the execution time scales very poorly with the amount of obstacles. This can be mitigated by only modeling a certain amount of obstacles relatively close to the vehicle, while limiting the vehicle to a convex region which does not overlap any of the ignored obstacles. Modeling this convex allowed region is very similar to modeling the obstacles, except that this time no integer variables are needed.

ACTIVE REGION

$$
\begin{aligned}
&minimize \quad N - \sum_{t=0}^{t \leq N} fin_t \\
&fin_0 = 1 \\
&fin_{t+1} = fin_t \vee cfin_{t+1}, \quad 0 \leq t < N \\
&cfin_{pos,t} = \bigwedge_{i=0}^{i<Dim(\boldsymbol{pos}_t)} |pos_{t,i} - pos_{goal,i}| < pos_{tol}, \quad 0 \leq t \leq N \\
&cfin_{vel,t} = \begin{cases} \bigwedge_{i=0}^{i<Dim(\boldsymbol{vel}_t)} |vel_{t,i} - vel_{goal,i}| < vel_{tol} & \text{if } \boldsymbol{vel}_{goal} \text{ exists, } 0 \leq t \leq N \\ true & \text{otherwise, } 0 \leq t \leq N \end{cases} \\
&cfin_t = cfin_{pos,t} \bigwedge cfin_{vel,t} \quad 0 \leq t \leq N
\end{aligned}
$$

6 Visualizing solution

black box solver means debugging is tricky. Need other way to get insight in what's happening. Visualization tool

- obstacles
- pre path

- corners
- path segments
- active region
- active obstacles w/ slack vars
- solver path
- exact value readout

7 Analysis and Results

7.1 Scenarios

7.2 Theta* vs A*

7.3 Bounding Box vs Genetic Algorithm

7.4 Curnercutting allowed vs not

7.5 Min/Max speed impact

7.6 Safety

7.7 Stability

7.8 Performance

7.9 Weaknesses

8 Conclusions

8.1 Future work

References

[1] John Saunders Bellingham. *Coordination and control of uav fleets using mixed-integer linear programming.* PhD thesis, Massachusetts Institute of Technology, 2002.

[2] Ian D Cowling, Oleg A Yakimenko, James F Whidborne, and Alastair K Cooke. A prototype of an autonomous controller for a quadrotor uav. In *Control Conference (ECC), 2007 European*, pages 4001–4008. IEEE, 2007.

[3] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 42–49. IEEE, 2015.

[4] Michele Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. Design of trajectory stabilizing feedback for driftless at systems. *Proceedings of the Third ECC, Rome*, pages 1882–1887, 1995.

[5] Melvin E Flores. *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational B-spline basis functions.* PhD thesis, California Institute of Technology, 2007.

[6] Yongxing Hao, Asad Davari, and Ali Manesh. Differential flatness-based trajectory planning for multiple unmanned aerial vehicles using mixed-integer linear programming. In *American Control Conference, 2005. Proceedings of the 2005*, pages 104–109. IEEE, 2005.

[7] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.

[8] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *Control Conference (ECC), 2001 European*, pages 2603–2608. IEEE, 2001.