Mixed Activity and Path Planning for Differentially Flat Systems

Enrique Fernandez[1]

***Abstract*— In my 6.832 Underactuated Robotics final project I focus on the problem of activity planning for dynamic underactuated systems. The result of this project is a planning framework that combines high level activity planning with path planning using bezier based smooth obstacle free paths. In this paper I describe the planning framework and show its application on a quadcopter package delivery scenario.**

I. Introduction

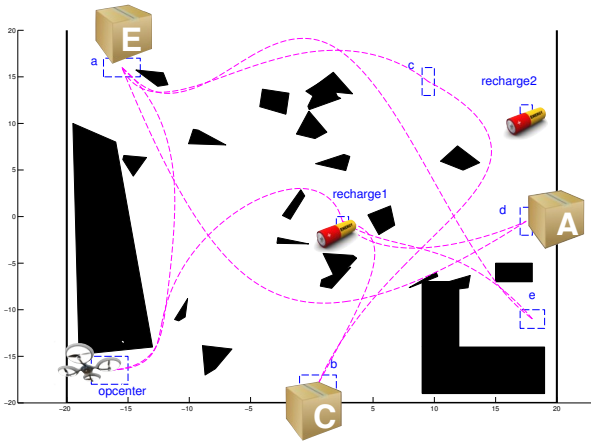
The field of activity planning has matured very significantly over the last few years. As demonstrated in the biannual International Planning Competitions, planners can now efficiently solve complicated planning problems in a reasonable time. Although classical planners have been extended to reason with time, metric resources and even a limited form of continuous effects, it is still rare to see planners dealing with both discrete actions and continuous path planning simultaneously. This project aims to close that gap by providing a framework for simultaneous activity planning and smooth path planning for underactuated robots in a scenario crowded with obstacles.

One of the goals of the project is to provide paths that are feasible given the dynamics of the underactuated robots used. In this project, I have chosen to assume that the dynamic systems allowed by the planning framework are differentially flat systems. The advantage of this assumption is that as long as the computed paths are sufficiently smooth, the paths will be dynamically feasible. Also, it means that we can limit the path planning to find the desired trajectory, as the inputs to the system to follow that trajectory can be found afterwards.

An example of differentially flat systems are those represented by the Dubins model:

$$\dot{x_D} = v_D cos(\phi) \tag{1}$$
$$\dot{y_D} = V_D sin(\phi) \tag{2}$$
$$\dot{\phi} = u_D \tag{3}$$
$$|u_D| \leq \frac{v_D}{\rho} \tag{4}$$

in which v_D is the fixed velocity and ρ is the minimum turning radius. A lot of systems can be represented with this simple model, such as a bicycle or a plane moving in 2D. The benefit of this representation for our problem is that the system will be able to follow any path as long as it is twice differentiable and respects the maximum curvature constraint. This reduces the planning problem to finding trajectories with those two properties.



(a) Planning scenario with obstacle-free smooth solution paths in magenta.

t(s)	Action	Duration (s)
0.0	(**navigate** quad0 loc-opcenter loc-recharge1)	114.4
114.4	(**recharge** quad0 loc-recharge1)	17.0
131.5	(**navigate** quad0 loc-recharge1 loc-b)	71.0
202.5	(**pickup** quad0 packet-bc loc-b)	2.0
204.5	(**navigate** quad0 loc-b loc-c)	138.6
343.2	(**deliver** quad0 packet-bc loc-c)	2.0
345.2	(**navigate** quad0 loc-c loc-a)	101.4
446.7	(**pickup** quad0 packet-ae loc-a)	2.0
448.7	(**navigate** quad0 loc-a loc-e)	202.5
651.2	(**deliver** quad0 packet-ae loc-e)	2.0
653.2	(**navigate** quad0 loc-e loc-recharge1)	69.8
723.0	(**recharge** quad0 loc-recharge1)	10.4
733.5	(**navigate** quad0 loc-recharge1 loc-d)	62.4
795.9	(**pickup** quad0 packet-da loc-d)	2.0
797.9	(**navigate** quad0 loc-d loc-a)	185.3
983.3	(**deliver** quad0 packet-da loc-a)	2.0
985.3	(**navigate** quad0 loc-a loc-opcenter)	128.7

(b) Solution plan for the example problem. **Navigate** actions have associated smooth obstacle-free paths, shown in magenta in (a)

Fig. 1: Quadcopter Package Delivery Problem. (a) shows the planning scenario, with obstacles in black, customer and recharge regions in blue and solution smooth obstacle free paths in magenta. (b) shows the activity schedule that indicates the delivery order, when to recharge and in what location.

A. Example planning problem

In this project, I will demonstrate my mixed activity and planning framework in a quadcopter package delivery scenario. Quadcopters are differentially flat systems (Mellinger and Kumar, 2011). Although quadcopters can follow paths with any curvature (they can always stop and hover), we still

*This paper is my final project for the Fall 2014 6.832 Underactuated Robotics MIT course submitted on December 17th, 2014.

[1]Enrique Fernandez is a PhD Candidate in the MERS lab at MIT CSAIL efernan@mit.edu

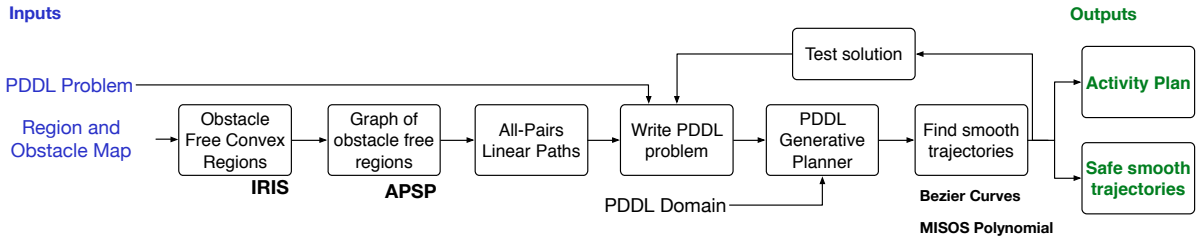


Fig. 2: Mixed activity and path planning framework.

want to generate smooth paths, so that the quadcopter can efficiently move without having to stop.

The planning problem consists on the following. A package delivery company owns a quadcopter that is initially in the Operations Center. The company has customers a, b, c, d and e, each denoted by a region in the map. The quadcopter has the mission of delivering some packages from customer to customer and returning to the operations center at the end (three packages in this example: $a \rightarrow e$, $b \rightarrow c$ and $d \rightarrow a$). Moreover, the quadcopter has limited battery life. Navigating from location to location consumes an amount of battery proportional to the length of the path followed. As the initial battery life of the quadcopter is not sufficient to finish the complete mission, there are two recharge points in the map in which the quadcopter can recharge its battery.

The quadcopter can perform discrete activities (*pickup, deliver, recharge*) and a continuous activity, *navigate*. The planning problem consists on:

- Finding the sequence of activities (schedule) that satisfies the plan goals while meeting the activities' preconditions. This involves:
  - Deciding the delivery order of the packages.
  - Deciding when to recharge and in which recharge point to ensure the quadcopter is able to finish the mission.
- Finding a smooth obstacle-free path for each navigate_A_to_B action.

Figure 1 shows the example planning problem. The planning scenario, with obstacles and customer and recharge regions can be seen in figure 1(a). Table 1(b) shows the solution schedule for this problem. This schedule contains the sequence of activities that the quadcopter needs to execute to solve the problem. This includes the delivery order, and when and where to recharge. Note that **navigate** activities have associated smooth obstacle-free paths (shown in magenta in figure 1(a)) that should be dynamically feasible.

The combined solution of the schedule and smooth obstacle-free paths is the output of the planning framework described in this paper. In the next sections I describe how that solution is found.

II. Approach Overview

The key idea for combining activity and path planning consists on using an off-the-shelf generative planner to obtain the schedule of activities to fulfill the mission. In this project I'm using the temporal planner COLIN (Coles et al., 2009). As most generative planners do, this planner takes a domain and problem specification written in PDDL and returns a solution schedule. The domain file specifies the activities that the quadcopter can execute (navigate, pickup, deliver and recharge) with their preconditions and effects. The problem file specifies the goals of the planning problem: what packages need to be delivered where, the initial battery level and the starting and final positions of the quadcopter.

The COLIN planner is not aware of the positions of the customer regions or the obstacles in the map and thus, does not perform path planning. Instead, the approach that I chose consists on reducing the **navigate** activity from one region to another to a discrete activity that doesn't consider obstacles. The way I do this is by first finding the minimum obstacle free linear trajectory lengths between all pairs of regions of interest (such as the customer regions or the recharge regions). These minimum distances are encoded in the problem PDDL file. The reason these distances are needed is because they affect the battery consumption while navigating from one region to another. The planer needs this information to determine whether the current battery level is sufficient to reach a region or if the quadcopter needs to recharge first. A related method for mixed activity and path planning has been developed in the context of planning AUV underwater inspection missions (Cashmore et al., 2014). In their approach, Cashmore et al. use probabilistic road maps to generate a graph of waypoints in the map and they annotate the distances between those waypoints. They then use a generative planner to find the sequence of waypoints that the AUV needs to visit to complete the mission. They however do not deal with the problem of generating paths that are not linear and they don't worry about the dynamic feasibility of the generated solution.

The mixed activity and path planning framework described in this paper is represented in figure 2. As mentioned above, the first step consists on obtaining all the pairwise minimum linear path lengths between regions of interest. For that, we first find convex obstacle free regions in the map using the IRIS algorithm (Deits and Tedrake, 2014b). We compute enough obstacle free convex regions to cover most of the map, ensuring all regions of interest are connected if possible. In the next step I find the connectivity graph of the obstacle free regions by finding all the pairwise intersections. Finding the all pairs shortest paths in this graph lets me find the shortest sequence of obstacle free regions that connects two regions of interest (for example to customer regions).

With this information, computing the shortest linear paths between regions of interest can be done using a quadratic program.

As mentioned earlier, the next step consists on calling the generative planner using the problem PDDL file that contains these all pairs minimum obstacle free distances. The result is the activity schedule that satisfies the mission.

The last step consists on finding smooth dynamically feasible trajectories for the quadcopter. We have already discussed that it is possible to find linear obstacle free paths. However, these have infinite curvature and are, therefore, not desirable. In the last step I compute a smooth obstacle free path for all **navigate** activities that appear in the solution schedule. In this project I have developed a method for finding these smooth obstacle free trajectories using Bezier curves. For comparison, I also tried to generate these trajectories using the mixed-integer sum of squares method developed for UAV planning in cluttered environments (Deits and Tedrake, 2014a).

Note that these smooth trajectories will, in general, be longer than the linear paths discussed earlier. Because of that, it is important to check if the proposed solution still holds, as these longer paths may be infeasible due to battery or temporal constraints not accounted for by the planner. If that is the case, the PDDL problem file can be rewritten with the new minimum distance information obtained at this step. The planner can then be called again and will now generate a plan that accounts for this more accurate estimate of the distance between those two regions. This iterative process can be repeated until the final solution is feasible. Testing the feasibility of the plans incorporating these smooth trajectories and replanning in case of inconsistencies was not done in this project due to time constraints, but its implementation is straightforward.

The rest of the document provides more details on how the different sections described in figure 2 work.

A. Generating obstacle free convex regions

In this section we generate convex obstacle free regions using the IRIS algorithm (Deits and Tedrake, 2014b). The IRIS algorithm is able to grow the largest convex region from a seed point that is guaranteed to be obstacle free. Because the generated regions are convex, we can easily ensure that a point is safe by making it satisfy the linear inequality constraints in the form of $A_i x \leq b_i$ for some safe region i. We use the algorithm described in (Deits and Tedrake, 2014a) to automatically find these regions. Figure 3 shows the first computed region and all safe regions for the example discussed in this paper.

B. Generating the graph of obstacle free convex regions

In figure 3 we can see how the safe obstacle free regions overlap. In this part of the planning framework we compute the connectivity graph of the random obstacle free regions calculated with the IRIS algorithm. Each node in the graph is a safe obstacle free region and edges between regions represent that they are connected. Since the regions

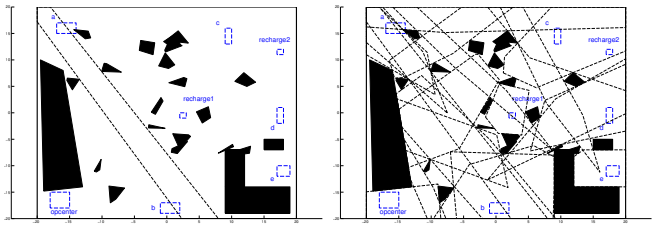


Fig. 3: IRIS regions (1 and 20 regions shown)

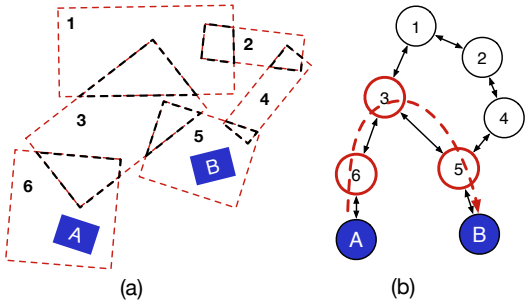


Fig. 4: Connectivity graph of safe regions

returned by the IRIS algorithm are polytopes, I use the Multi-Parametric Toolbox (Herceg et al., 2013) to check all pairwise intersections between the safe regions. For each pair of regions whose intersection is non-empty, a bidirectional edge is added to the graph. Those intersections and the resulting connectivity graph are represented in figure 4.

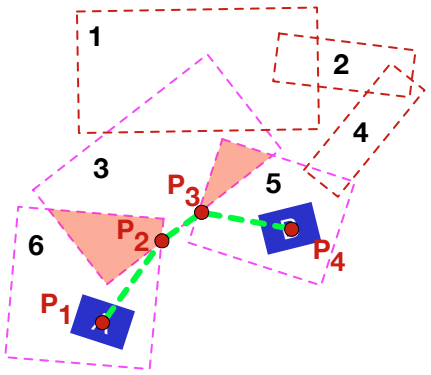
In the next step, I compute the all-pairs-shortest-paths (APSP) on the graph using the *Floyd-Warshall* algorithm (Wikipedia, 2014b). This provides the shortest paths between any two safe regions. The shortest path is understood here as the path that goes through the smallest possible number of intermediate safe regions. Then each region of interest (customer or recharge regions, for example) are associated to a safe region. In this project this is done, again, by checking the intersection between all safe regions and each region of interest. The safe region assigned to each region of interest is the one whose intersection with the region of interest has the largest volume.

We can now compute the 'safe region path' between any two regions of intersest. In the example shown in figure 4, we can see how the *best path* from region of interest A to region of interest B is determined to be the one that goes through safe regions 6, 3 and 5 (in that order).

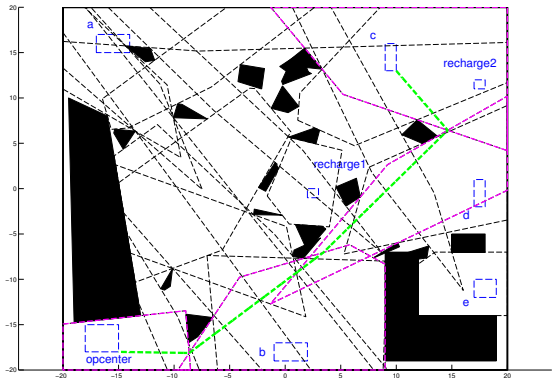
C. Finding the all pairs minimum linear distances

Using the approach explained in the previous section we can find the *safe region path* between any two regions of interest. We can use this information to find the shortest linear obstacle free path going through that sequence of safe regions. I do it by formulating a *Quadratic Program* that is solved using Yalmip (Löfberg, 2004) and the Gurobi solver.

The linear path we are looking for consists of as many linear segments as safe regions contained in the *safe region path*, denoted as n. There are, therefore, $n+1$ points that



(a)



(b)

Fig. 5: Linear paths are found using Quadratic Programming

connect these segments. The objective of the Quadratic Program is to find the coordinates of those $n+1$ points. The key insight to ensure that the linear path is obstacle free is to make each of the segments be fully contained in one of the safe regions. To do that, we only need to impose the condition that all the points (except for the initial and last one) be contained in **both** the previous and the current safe region in the sequence. Because the safe regions are convex, all the segments are then fully contained in a safe region and thus the path is obstacle free. To obtain the shortest linear path, we can minimize the sum of the squared norm of each of its segments as expressed with the following equations.

$$\begin{aligned} \min_{x} \quad & \sum_{i=2}^{n} ||P_i - P_{i-1}||^2 \\ \text{s.t.} \quad & A_{i-1}P_i \leq b_{i-1} \qquad \forall i \in 2,\ldots,n-1. \\ & A_iP_i \leq b_i \qquad \forall i \in 2,\ldots,n-1. \end{aligned} \tag{5}$$

where the *safe region path* is given by the sequence of regions $i = 1,\ldots,n$, convex polytopes represented by the linear inequalities $A_i x \leq b_i$.

The first and the last points can either be chosen freely inside the regions of interest, or be completely specified (such as the center of each region of interest). An example of this quadratic program can be seen in figure 5(a). As explained earlier, the goal is to place points P_2 and P_3 in their shaded regions such that the linear path is shortest.

The algorithm explained above is applied to all pairs of regions of interest to find their shortest linear distances. Figure 5(b) shows an example of a linear path computed between two regions of interest in the scenario discussed in this paper.

Note that the linear obstacle free paths discussed in this section are, in general, not the shortest possible paths given the obstacles in the map. There are two reasons for this: a) the obstacle free convex regions computed with the IRIS algorithm don't cover the complete obstacle free area of the map; and b) the fact that we choose the *safe region path* with the smallest number of regions doesn't ensure that the linear path through does regions is shorter than a different one going through more regions, but better arranged.

D. Finding the plan with a generative planner

In this step we ask the generative planner, COLIN, to find the activity plan and schedule for us. The inputs to COLIN are the domain PDDL file and the problem PDDL file. The domain file describes the activities that the quadcopter can execute, such as picking up a package, recharging, or navigating from one region to another. These activities are parameterized on PDDL objects. For example, the **navigate** activity is parameterized on the start and goal region of interest. While the navigate activity is only defined once in the domain file, the planner grounds this action for all possible combinations of start and goal regions of interest so that *navigate_a_to_b*, *navigate_e_to_c* and all others exist.

The pddl for the **navigate** activity is shown below.

```
(:durative-action navigate
:parameters (?x - quadcopter ?y - location ?z - location)
:duration (>= ?duration (mintime ?y ?z))
:condition (and
            (over all (can_traverse ?x ?y ?z))
            (at start (available ?x))
            (at start (at ?x ?y))
            (at start (>= (battery ?x) (cost ?y ?z)))
            )
:effect (and
              (at start (not (at ?x ?y)))
              (at end (at ?x ?z))
              (at start (moving ?x ?y ?z))
              (at end (not (moving ?x ?y ?z)))
              (at end (decrease (battery ?x) (cost ?y ?z)))
              ))
```

The **navigate** activity takes the quadcopter from starting location ?y to end location ?z given that the requirements are satisfied. These requirements, among others, indicate that the quadcopter has to be in region ?y before executing the activity. More importantly, this activity requires that the battery level of the quadcopter is greater than the **cost** to reach region ?z from region ?y.

These costs are proportional to the shortest linear obstacle free path lengths calculated in the previous region, and are specified in the problem PDDL file with statements such as:

```
(can_traverse quad0 loc-a loc-b) (= (cost loc-a loc-b) 19.988)
(can_traverse quad0 loc-b loc-a) (= (cost loc-b loc-a) 19.988)
(can_traverse quad0 loc-a loc-c) (= (cost loc-a loc-c) 16.336)
(can_traverse quad0 loc-c loc-a) (= (cost loc-c loc-a) 16.336)
```

The planner needs this information to determine whether it can reach a region from another one with the given battery level or if it needs to recharge on the way.

The problem PDDL file also specifies the available customer and recharge regions, the starting location and battery level of the quadcopter, the location of the packages and where they need to be delivered and more.

The resulting plan generated by COLIN for the example scenario is the one shown in table 1(b). The plan specifies that the quadcopter needs to recharge at recharge location 1, then deliver the package in b to c, then deliver the package in a to e, then recharge again in recharge location 1, then deliver the package in d to a and, finally, navigate to the operations center. While COLIN ensures the correctness of the plan with respect to the requirements specified in the PDDL files, there are no guarantees of optimality with respect to the number of activities executed, minimum execution time, or any other. This is common in generative planners as they often use greedy search methods.

E. Generating smooth obstacle free trajectories

At this point, we now have a feasible plan if we consider that the quadcopter can follow linear paths. However, these paths have infinite curvature and would require the quadcopter to hover at the end of each segment, orient itself to the new direction and continue along the next segment. While this is feasible for a quadcopter, this won't be possible for other systems such as the ones represented with the constant velocity Dubins model introduced at the beginning of this paper. Therefore, the last step of the planning framework consists on extracting the **navigate** activities from the plan and generating smooth obstacle free paths from the starting region of interest to the final region of interest for each of them.

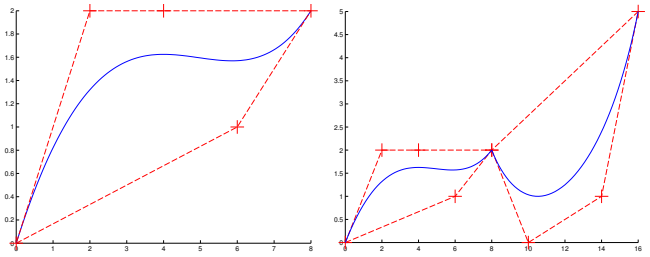
In the solution plan for the example problem shown in table 1(b) there are 9 navigate activities, such as `(navigate quad0 loc-b loc-c)`, that need a smooth obstacle free trajectory. In the next section of this paper I explain how to obtain these trajectories using Bezier curves.

Note that the length of the smooth trajectories will be in general larger than the length of the linear trajectories. Because of this, the plan generated by COLIN may no longer be feasible (a navigate activity along the smooth path may use more battery than available). Although this is ignored in this project due to timing constraints, I discussed in the approach overview section of this paper that this issue can be tackled by testing the validity of the plan and iteratively updating the cost of the navigate actions and solving for new plans with COLIN until a valid one is discovered.

III. Smooth obstacle free trajectories with Bezier curves

A Bezier curve of degree n is defined by its $n+1$ control points: $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$, according to the following formula:

$$\mathbf{P}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P}_i, \quad t \in [0,1] \tag{6}$$



(a) (b)

Fig. 6: 4th degree Bezier curve in (a). In (b), same curve connected with a cubic Bezier curve.

where $B_i^n(t)$ is the Bernstein polynomial of degree n:

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \tag{7}$$

Bezier curves have interesting properties that are very beneficial for planning obstacle free paths. The curves start and end in the first and last control points and they are fully contained in the convex hull of their control points. That means that if we place all the control points in one of the safe regions calculated in a previous section, the resulting Bezier curve is guaranteed to be contained in that safe region and, therefore, obstacle free.

A Bezier curve of 4th degree is shown in figure 6(a). The five control points are shown with red crosses. Note that the curve is contained in the convex hull of the control points. The algorithm to find a smooth obstacle free path from a region of interest to another consists on piecing as many bezier curves as the number of safe regions in the *safe region path*. Each of the bezier curves will be fully placed in its corresponding safe region from the *safe region path*. The problem, then, consists on finding the control points of each of the bezier curves.

The control points need to be chosen so that the overall path is C^2 continuous (twice differentiable). This condition ensures that the path has continuous curvature. The conditions on the control points for a Bezier curve (curve i) and the previous one ($i-1$) are the following (Choi, Curry, and Elkaim, 2010):

$$ {}^{i-1}P_{n_{i-1}} = {}^{i}P_0, \tag{8}$$

$$ n_{i-1}({}^{i-1}P_{n_{i-1}} - {}^{i-1}P_{n_{i-1}-1}) = n_i({}^{i}P_1 - {}^{i}P_0), \tag{9}$$

$$ n_{i-1}(n_{i-1}-1)({}^{i-1}P_{n_{i-1}} - 2\,{}^{i-1}P_{n_{i-1}-1} + {}^{i-1}P_{n_{i-1}-2}) \tag{10}$$

$$ = n_i(n_i-1)({}^{i}P_2 - 2\,{}^{i}P_1 + {}^{i}P_0) \tag{11}$$

These are linear constraints and come from the derivation rules for Bezier curves. As discussed earlier, the control points of each Bezier segment also need to be inside of their assigned safe region to guarantee that the curve won't collide with any obstacles. Choosing the first control point of the first Bezier segment and the last control point of the last Bezier segment to be in the start and end regions of interest

and enforcing the previous continuity constraints ensures that the Bezier path is smooth, obstacle free and takes from the start region of interest to the goal region of interest.

We now need to decide the objective function to minimize in order to choose the control points. We want to obtain paths that are smooth (low curvature) and whose length is relatively small. I experimented with multiple objectives, but in the end the one that provided the best results was a combination of the integral of the snap and the squared length of the curve.

$$\min_{{}^1\mathbf{P}_0,\ldots,{}^n\mathbf{P}_{n_n}} \quad k_S \int_0^T \left(\frac{d^4x(t)}{dt^4}\right)^2 + \left(\frac{d^4y(t)}{dt^4}\right)^2 dt + k_L \text{length}^2 \tag{12}$$

The justification for minimizing the integral of the snap (fourth derivative) is that paths resulting from this minimization seem to match well human preferences (Mellinger and Kumar, 2011). Since Bezier curves are polynomial curves, the integral of the norm squared snap can be found analytically as a quadratic function on the control points. This is done by finding the polynomial coefficients of the Bezier curve as a function of the control points (Wikipedia, 2014a). The length of a Bezier curve can't be found analytically, instead, I use an approximation based on Legendre-Gauss Quadrature (Kamermans, 2014). This length approximation can also be written as a quadratic function on the control points. In my tests, the values of $k_S = 20$ and $k_L = 1$ seemed to work well.

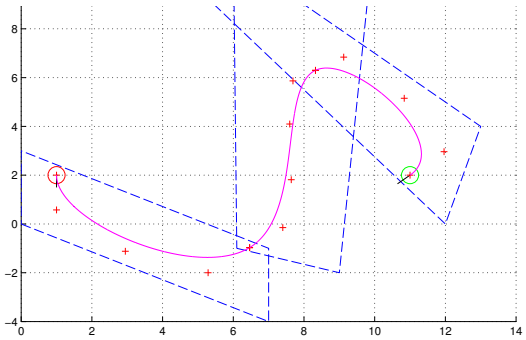


Fig. 7: Bezier path composed of three segments. Control points are shown in red crosses.

The problem formulated above with a quadratic minimization objective and linear constraints for continuity, start and end conditions and making the control points be inside the convex free regions is a *Quadratic Program*. As in the linear paths case, the QP was solved using Yalmip and the Gurobi solver. For each of the paths calculated for the example scenario, Gurobi typically found the solution in 2-3 seconds. Figure 7 shows a Bezier path composed of three segments going from the red circle to the green circle. In this example the safe regions are the three polytopes plotted in dashed blue lines. In this case, the three bezier curves have degrees 5, 6 and 5. After experimenting with several values, I decided to use fourth degree curves in the example scenario.

Figure 8 shows the Bezier paths generated for the first four navigate activities in the example scenario. Note that the

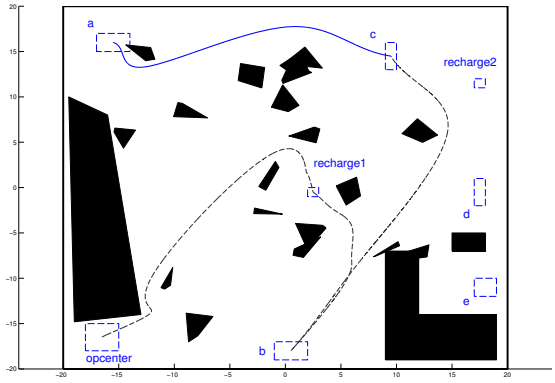


Fig. 8: Bezier paths generated for the first four navigate activities in the example scenario. The rest are not show to preserve clarity

paths are smooth and obstacle free. Although the results are pretty good, some of the paths present excessive curvature, resulting in unnatural and, in some cases, difficult to follow paths. I believe the reason for this is that the way I formulate the optimization problem, all bezier segments are supposed to have the same duration, even though some are much longer than others. Because that is not properly taken into account in the minimization objective, I believe the real snap is not minimized and, thus, the curvature in some areas is bigger than it should be.

In this project I also tried to compute smooth obstacle free paths using the mixed integer second-order cone (MISOCP) approach described in (Deits and Tedrake, 2014a). I adapted the code from that paper to work in my 2D scenario. This approach tries to fit m segments of nth order polynomials to get from the start to the goal while ensuring that each of the polynomials is fully contained in one of the convex safe regions. I tried computing the paths using 10 segments of 3rd order polynomials for each of the navigate activities. The original algorithm uses mixed integer programming to assign each of the polynomial segments to any of the safe regions in the map. In order to improve the computation time, in my tests I limited the safe regions that the algorithm could choose for each of the polynomial segments to only the safe regions in the *safe region path* whose computation was described in an earlier section of this paper. The results are shown in figure 9.

As shown in the figure, the paths generated using this method are very smooth and natural. The only problem with this approach is that the computation time took orders of magnitude longer than the Bezier curve approach described in this section.

One of the reasons why the MISOCP method takes longer is probably because it is doing a mixed integer optimization as it is trying to assign each of the polynomial segments one of the safe regions in the *safe region path*.

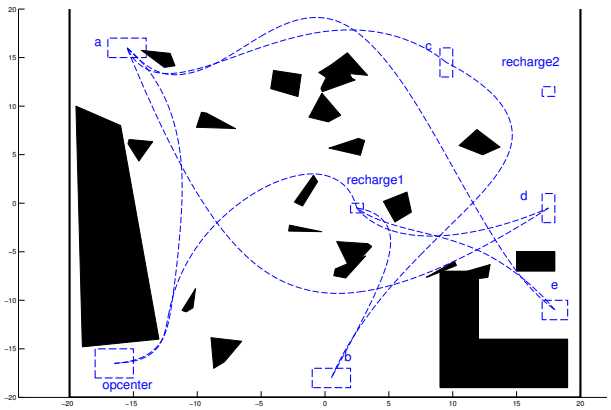


Fig. 9: Smooth cubic polynomial paths computed using the MISOCP approach (Deits and Tedrake, 2014a)

Trajectory	Bezier (s)	MISOCP (s)
(**navigate** quad0 loc-opcenter loc-recharge1)	3.15	154.39
(**navigate** quad0 loc-recharge1 loc-b)	2.60	99.84
(**navigate** quad0 loc-b loc-c)	2.06	34.20
(**navigate** quad0 loc-c loc-a)	2.09	35.50
(**navigate** quad0 loc-a loc-e)	2.70	84.87
(**navigate** quad0 loc-e loc-recharge1)	3.36	173.73
(**navigate** quad0 loc-recharge1 loc-d)	2.19	53.27
(**navigate** quad0 loc-d loc-a)	2.81	83.25
(**navigate** quad0 loc-a loc-opcenter)	2.85	69.44
TOTAL	23.81	788.49

TABLE I: Computation time of the Bezier and MISOCP methods for each of the curves in the example scenario. The Bezier method is more than an order of magnitude faster than the MISOCP method.

IV. Future Work

There are many things I would like to do to improve this project. In first place I would like to fully implement the iterative planning approach that ensures that the smooth trajectories calculated at the end still satisfy the plan generated by the generative planner. As I mentioned in an earlier section, the final smooth paths should be checked to see if they satisfy the battery constraints. If that's not the case, the minimum cost for the relevant navigate from A to B activity should be updated and the generative planner should find a new plan. The process would stop once the final computed paths satisfied all the constraints.

Another thing I would like to do is fix the excessive curvature problems that I found in some cases using the Bezier curve approach. As mentioned in that section, the error is most likely due to the fact that all Bezier trajectory segments are considered to have the same duration, and therefore the computed snap is not exactly the correct one. But even more important than fixing that problem would be to find a way to ensure that the path respects some maximum curvature constraint. If this was the case, we could compute trajectories that systems described by models such as the Dubin's one explained at the beginning of the paper could traverse. In this project I tried to limit the maximum curvature of the path, described by the following formula:

$$\kappa(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{(\dot{x}^2(t) + \dot{y}^2(t))^{\frac{3}{2}}} \tag{13}$$

Unfortunately equation 13 is highly non-linear. I tried setting constraint to limit the curvature at small discretized steps along the path and solving the overall problem with SNOPT. Although SNOPT returned an optimal solution pretty fast, the generated curves were not satisfactory. The problem is that while the curvature constraints were satisfied at the discretized points, the curvature acquired very high values out of those points, making the path look unnatural and hard to follow.

Another thing that I would have liked to do in this project is to, also, calculate smooth paths using *trajectory optimization*. The benefit of this approach is that the generated paths would be feasible for the studied system by definition. The maximum curvature constraint would be embedded in the dynamics of the system and respected in the final path. I think the hardest part would be to encode the obstacles (or obstacle free safe regions) using an efficient formulation that didn't complicate the problem too much. The problem could be formulated trivially using a mixed-integer non-linear program (MINLP), but I believe this approach would be computationally expensive. Moreover, there are not many efficient MINLP solvers, and that makes the problem even harder. Different formulations not involving integer variables are possible, but not straightforward.

V. Conclusion

At the beginning of this project I set the goal of doing a project on activity planning for dynamic underactuated systems. My daily research is on activity planning, but, in the beginning, I really didn't know how I was going to achieve my objective. It was reading (Deits and Tedrake, 2014a) and (Deits and Tedrake, 2014c) that introduced me to the IRIS algorithm and gave me inspiration to frame this problem. Working on this project has been very time-consuming, but very rewarding as well, and I believe I have accomplished the goal I set in the beginning. The 6.832 Underactuated Robotics class has been a really good class and a changing experience for me. I believe what I've learn in this class and the inspiration from the topics and other people's projects are going to have a great impact in my research at MIT going forward.

VI. Software

All the MATLAB code used for this project is available in Github (`https://github.com/enriquefernandez/mixed_activity_path_planning`).

References

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. AUV mission control via temporal planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 6535–6541.

Choi, J.-w.; Curry, R. E.; and Elkaim, G. H. 2010. Curvature-continuous trajectory generation with corridor constraint for autonomous ground vehicles. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, 7166–7171.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2009. Temporal Planning in Domains with Linear Processes . *... of the International Joint Conference on ...*.

Deits, R., and Tedrake, R. 2014a. Efficient Mixed-Integer Planning for UAVs in Cluttered Environments. *groups.csail.mit.edu*.

Deits, R., and Tedrake, R. 2014b. Computing Large Convex Regions of Obstacle-Free Space through Semidefinite Programming. In *Proceedings of the Eleventh International Workshop on the Algorithmic Foundations of Robotics (WAFR 2014)*, 1–16.

Deits, R., and Tedrake, R. 2014c. Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization. In *Proceedings of the 2014 IEEE/RAS International Conference on Humanoid Robots (Humanoids 2014), Madrid, Spain, 2014*.

Herceg, M.; Kvasnica, M.; Jones, C. N.; and Morari, M. 2013. Multi-Parametric Toolbox 3.0. In *Proc of the European Control Conference*, 502–510.

Kamermans, M. 2014. A Primer on Bézier Curves.

Löfberg, J. 2004. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proceedings of the CACSD Conference*.

Mellinger, D., and Kumar, V. 2011. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2520–2525.

Wikipedia. 2014a. Bézier curve — Wikipedia, The Free Encyclopedia. Technical report.

Wikipedia. 2014b. Floyd–Warshall algorithm — Wikipedia, The Free Encyclopedia. Technical report.