Indian Institute of Technology,
Bombay

BTech Project

Motion Planning and Control of Two-Dimensional Aerial Robot

Author:
Savyaraj Deshmukh

Guide:
Prof. Vivek Sangwan

May, 2019

Contents

1 Introduction

In the recent years, there has been a lot of work in developing trajectories and controllers for quadcopters to perform variety of aggresive maneuvers [1] [2] [3] [4]. Authors in [1] successfully achieved maneuvers such as flying through narrow, vertical gaps and perching on inverted surfaces. They developed the dynamic models based on first principles and feedback control laws. In [2], trajectory generation algorithm was developed for a multirotor system to guide the vehicle from any initial state to any end state in the given amount of time.

In the last phase of this project, controllers were developed for a 2-dimensional quadcopter robot which is controlled by two thrusts. Given a trajectory $[x, y(x)]$, the task was to follow the trajectory with a given desired velocity along its tangent. For this purpose, cascade PID controllers were developed with linear as well as nonlinear dynamical models.

In this work, we have focused on the task of actually generating the trajectories in 2D space which can be used to maneuver the bot through obstacles. We consider a disturbance free environment that consists of planar obstacles and our goal is to take the bot from start position to the region of interest without colliding. The problem is formulated as an optimal control problem with the objective of generating minimum-time trajectories that are dynamically feasible for the bot. We follow the approach by [**?**] where motion primitives are used to generate trajectory sequences and an graph search algorithm called 'A* algorithm' is employed to find an optimal sequence of primitives. To accelerate planning, we first perform a lower dimensional search and use it as a heuristic to guide the generation of a final dynamically feasible trajectory.

2 Summary of Phase I

2.1 System Description

We begin with an environment that is a 2-dimensional space without any stochastic disturbances such as wind. Standard gravitational force is considered with acceleration due to gravity g and a drag force with a constant drag coefficient c. System consists of a 1-dimensional bot of constant length L which is controlled by two thrusts at its end T_l and T_r which act perpendicular to the bot length. The position and orientation of the bot is given by $[x, y, \theta]$ where x and y are its locations and θ is assumed be its angle measured clockwise from the positive X-axis. It's velocities are denoted as $[v_x, v_y, w]$.

2.2 Dynamics

We consider the bot as a uniform rigid body of mass M and moment of inertia I about its center. Its dynamical equations can be set up as (Figure 1):

$$M\ddot{x} - (T_l + T_r)sin\theta + c\dot{x} = 0$$
$$M\ddot{y} - (T_l + T_r)cos\theta + c\dot{y} - Mg = 0$$
$$M\ddot{\theta} - (T_l - T_r)\frac{L}{2} + c\dot{\theta} = 0$$

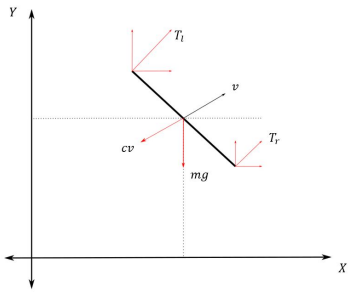

Figure 1: Bot

We have $v_x = \dot{x}$, $v_y = \dot{y}$, $w = \dot{\theta}$
Let $T_a = T_l + T_r$, $T_s = T_l - T_r$
We get

$$M\ddot{x} - T_a sin\theta + cv_x = 0$$
$$M\ddot{y} - T_a cos\theta + cv_y - Mg = 0$$
$$M\ddot{\theta} - T_s\frac{L}{2} + cw = 0$$

2.3 Control

The basic approach is to develop two controllers and composition of these two controllers (Figure 2). Each controller is tuned separately to achieve the desired performance characteristics. These controllers are described in the following schematic:

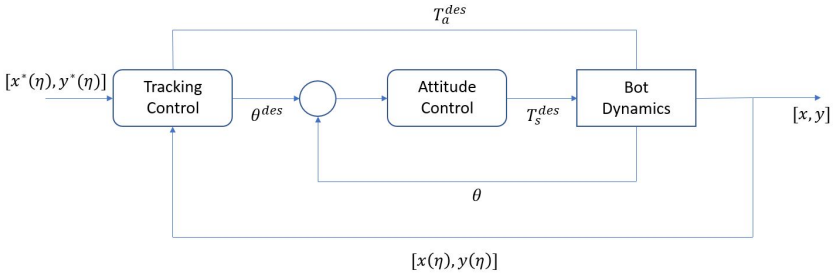

Figure 2: Controller Schematic

2.3.1 Cascade PID Control Scheme

1. *Attitude Control*
   Driving the bot to a desired orientation specified by θ while keeping a constant nominal thrust to keep it afloat

$$T_s = k_{p_\theta}(\theta^{des} - \theta) + k_{d_\theta}(\dot{\theta}^{des} - \dot{\theta})$$

2. *Trajectory Tracking Control*
   Controlling the centre of bot to follow a particular path in 2D space parametrized by $[x(\eta), y(\eta)]$ where η is a path co-ordinate, along with a specified velocity profile ${v_t}^*(\eta)$ along the path in the tangential direction.

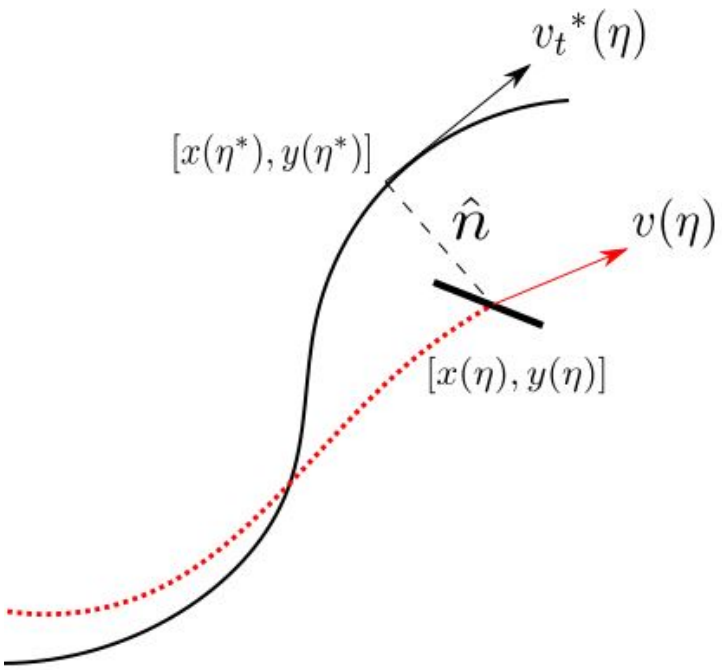

Figure 3: Tracking Errors

2.4 Thrust calculations

While calculating the rotor thrusts, we can approximate the dynamics using small angle approximation:

$$sin(\theta) \approx \theta, \qquad cos(\theta) \approx 1$$

2.4.1 Linear Controller

$$M\ddot{x} = T_a\theta$$
$$M\ddot{y} = T_a - Mg$$

$${T_a}^{des} = M(\ddot{y}^{des} + g)$$
$$\theta^{des} = \frac{M\ddot{x}^{des}}{{T_a}^{des}}$$

2.4.2 Nonlinear Controller

$$M\ddot{x} = T_a sin\theta$$
$$M\ddot{y} = T_a cos\theta - Mg$$

$$\theta^{des} = \tan^{-1}(\frac{\ddot{x}}{\ddot{y}+g})$$
$$T_a^{des} = M\sqrt{\ddot{x}^2 + (\ddot{y}+g)^2}$$

2.5 Results

To visualize the bot, a 2D figure was created in MATLAB. Bot is represented by a straight line of length L. Blue lines in the figures show desired trajectories and red dotted lines show the actual trajectories followed by the bot. The controller was based on nonlinear dynamical model of the bot.

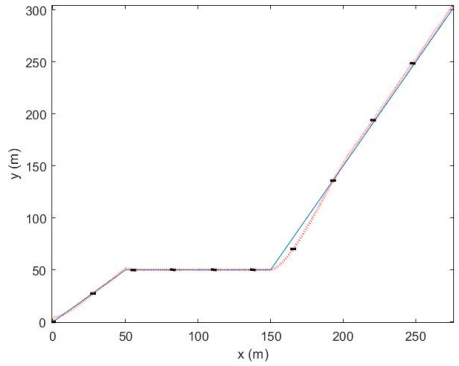

Figure 4: Piecewise Linear Trajectory

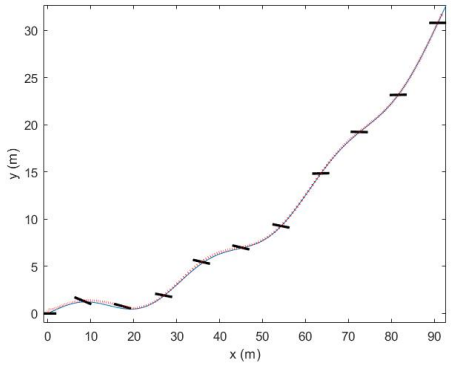

Figure 5: Irregular Trajectory

3 Motion Planning

The task of motion planning is tackled using motion primitives to discretize the control space and obtain a dynamically feasible resolution-complete (i.e., optimal in the discretized space) trajectory in environments with obstacles. Furthermore, to reduce computational efforts of finding optimal trajectories in high dimensional spaces, a new hierarchical planning process is introduced that refines a dynamically feasible trajectory from a prior trajectory in lower dimensional space.

Particularly, we built the following

- A graph search algorithm that uses motion primitives to compute a dynamically feasible resolution-complete trajectory that is optimal in that discretized space
- A hierarchical refinement process that uses prior lower dimensional trajectories as heuristics to accelerate planning in higher dimensions
- The effect of motion primitive discretization parameters on the computation time, smoothness, and optimality of the generated trajectories is analyzed

3.1 Motion Primitives

The desired trajectory can be defined as:

$$\Phi(t) := \left[\mathbf{x}^\top, \dot{\mathbf{x}}^\top, \ddot{\mathbf{x}}^\top, \dddot{\mathbf{x}}^\top\right]^\top = \left[\mathbf{x}^\top, \mathbf{v}^\top, \mathbf{a}^\top, \mathbf{j}^\top\right]^\top \tag{1}$$

The components of $\Phi(t)$ are represented as polynomial trajectories in time

$$\mathbf{x}(t) := \sum_{k=0}^{K} \mathbf{d}_k \frac{t^k}{k!} = \mathbf{d}_K \frac{t^K}{K!} + \ldots + \mathbf{d}_1 t + \mathbf{d}_0 \tag{2}$$

Such a trajectory defined for a specific amount of time is called a ***Motion Primitive***. To generate such primitives, we assume a 'control space', given by $u \in U = [u_1, u_2, \ldots u_n]$ applied for a small amount of time $\tau > 0$ that can be integrated to obtain different motion primitives.

Consider a specific example where our control space is characterized by jerk ($\dddot{x}$). We consider the jerk space $j \in J = [-j_{max}, -j_{max} + \delta j, \ldots j_{max}]$ where j_{max} is the maximum attainable jerk constrained by dynamical considerations of the bot. Here, the state is defined as:

$$\mathbf{s}(t) := \left[\mathbf{x}(t)^\top, \dot{\mathbf{x}}(t)^\top, \ddot{\mathbf{x}}(t)^\top\right]^\top = \left[\mathbf{p}^\top, \mathbf{v}^\top, \mathbf{a}^\top\right]^\top \tag{3}$$

we get the motion primitive for time duration $[0, \tau]$ as

$$\mathbf{s}(t) = F\left(\mathbf{u}_m, \mathbf{s}_0, t\right) := \begin{bmatrix} \mathbf{u}_m \frac{t^3}{6} + \mathbf{a}_0 \frac{t^2}{2} + \mathbf{v}_0 t + \mathbf{p}_0 \\ \mathbf{u}_m \frac{t^2}{2} + \mathbf{a}_0 t + \mathbf{v}_0 \\ \mathbf{u}_m t + \mathbf{a}_0 \end{bmatrix} \tag{4}$$

4 Graph Search

The finite control input set U_m and duration τ define a graph $G(S, E)$, where S is the set of reachable states in $\mathbb{R}^9$ and E is the set of edges connecting those states. The states in S are generated by applying each element of U_m at each state iteratively, and each element in E is a motion primitive as defined by (4). [**?**]

Our goal is to find a trajectory from a start state s_0 to goal state s_g which minimizes a cost functional J given by

$$\Phi^*(t) = \arg\min_{\Phi(t)} J + \rho T = \arg\min_{\Phi(t)} \int_0^T \|\mathbf{j}\|^2 dt + \rho T$$

Hence, for each motion primitive of duration τ, the cost function will be given by

$$C\left(\mathbf{s}_n, \mathbf{u}_m\right) = \left(\|\mathbf{u}_m\|^2 + \rho\right)\tau \tag{5}$$

It can be shown by Pontryagins minimum principle that (5) is the optimal solution of (4) [**?**]. Hence, we can redefine the original problem in continuous control space to the following one:

Problem Statement: Given an initial state s_0, obstacle free space R_{free} and a target region R_{goal}, we choose a sequence of motion primitives $u_m \in U_m$ for $t \in [0, \tau]$ such that

$$\begin{aligned} &\min_{N, \mathbf{u}_0:N-1} \left(\sum_{n=0}^{N-1} \|\mathbf{u}_n\|^2 + \rho N\right)\tau \\ &F_n(t) := F\left(\mathbf{u}_n \mathbf{s}_n, t\right), \quad \mathbf{u}_n \in \mathcal{U}_M \\ &\mathbf{s}_{n+1} = F_n(\tau) = F_{n+1}(0), \quad \mathbf{s}_N \in \mathcal{X}^{goal} \\ &F_n(t) \subset \mathcal{X}^{free} \end{aligned} \tag{6}$$

4.1 Costs and Heuristic Function

We use a graph search algorithm called 'A* algorithm' which is based on an heuristic $H(s_n, s_g)$ and a cost function $C(s_0, s_n, u_m)$. The algorithm attempts to minimize the weighted sum of current cost of the path and an lower bound for the estimate of the cost required to follow the path. Specifically, it minimizes effective cost function $F(s_n, s_g, U_m)$

$$F(s_n, s_g, U_m) = H(s_n, s_g) + C(s_0, s_n, u_m)$$

4.1.1 Cost

The cost of each primitive is the time taken to complete that primitive, τ. As we are interested in smooth and dynamically feasible trajectories, control costs are also taken into account. Hence, the total cost for a single primitive becomes

$$C\left(\mathbf{s_n}, \mathbf{u}_m\right) = \left(\|\mathbf{u}_m\|^2 + \rho\right)\tau \tag{7}$$

Now, at any state s_n, the total cost function $C(s_0, s_n, u_m)$ is

$$C(s_0, s_n, u_m) = \sum_{i=0}^{n} C(s_i, u_m), \qquad s_i = F(s_{i-1}, u_m^*, t) \tag{8}$$

where u_m^* correspond to the motion primitives that give minimum cost up to the state s_n.

4.1.2 Heuristic

A heuristic function $H(s_n, s_g)$ gives an underestimate of the cost to travel from the state s_n to goal s_g. It gives an idea of how 'close' or 'far away' the goal is from the current state. One of the easiest ways in this case is to ignore the control cost and calculate the lower bound on total time taken from $s_n \rightarrow s_g$. Assuming v_{max} as the maximum feasible velocity for the bot in each x and y directions, this can be estimated as

$$H\left(s_n, s_g\right) = \rho \frac{\|p_n - p_g\|}{v_{\max}}$$

This function has been used for the trajectory planning in the velocity space.

For higher dimensional spaces, it was observed that naive approximation of the minimum time cost was not sufficient for smooth and computationally efficient generation of trajectories. Hence, we have introduced an extra term in the heuristic that takes into account the velocities attained by the bot. Let a_{max} as the maximum feasible acceleration for the bot in each x and y directions

$$H\left(s_n, s_g\right) = \rho \frac{\|p_n - p_g\|}{v_{\max}} + \rho * c * \frac{\|v_n - v_g\|}{a_{\max}}$$

where c denotes the relative significance of both these costs.

Now, it may happen that this heuristic fails to be an underestimation of the actual cost, a necessary condition for the optimality of A* algorithm. But this new term was very useful to find feasible and smooth trajectories.

5 Motion Feasibility

We need to consider both dynamical constraints that arise from system dynamics and geometric constraints due to physical obstacles.

5.1 Collision Avoidance

Here, the bot is modelled as a rectangle that encloses the actual shape 6. To check if a motion primitive is violating any obstacle constraints, the trajectory is divided into I states and the primitive is considered to be collision free if

$$\mathcal{O} \cap \xi\left(\mathbf{s}_{i,n}\right) = \varnothing, \forall i = \{0, 1, \ldots, I-1\}$$

where $\mathcal{O}$ denotes the set of obstacles, $\xi\left(\mathbf{s}_{i,n}\right)$ are the trajectories from $(\mathbf{s}_{i,n})$ states that are sectioned from the s_n state.

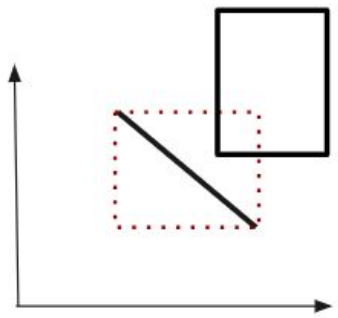

Figure 6: Bot model for collision avoidance

5.2 Dynamic Feasibility

Dynamic feasibility of the trajectory comes from the limitations of motor torques and their response times. It is also necessary to limit bot velocities to achieve better stability and control over the bot. Utilizing the property of differential flatness of this system, we can use the same constraints in different axes independently. They are given by

$$|\dot{\mathbf{x}}(t)| \preceq \mathbf{v}_{\max}, \quad |\ddot{\mathbf{x}}(t)| \preceq \mathbf{a}_{\max}, \quad |\dddot{\mathbf{x}}(t)| \preceq |\mathbf{j}_{\max}| \tag{9}$$

During the motion planning, we check for above constraints only at the states that are calculated so far. Although checking for the constraint at intermediate points or a closed form solution of the maximas in state variables might give more accurate results, we observed that checks only at states was more than sufficient to ensure feasibility and computationally much more advantageous.

6 Results

6.1 Parameters

Dynamic constraints and discretization scheme demands various parameters to be defined. Velocity and acceleration constraints are based on real limitations of the equivalent quadcopter. While choosing discretization parameters, there is trade-off between computational efficiency and accuracy.

ρ	τ	v_{max}	a_{max}	j_{max}	bin size
$15{*}u_m^2$	0.2 s	7 m/s	$7m/s^2$	20 m/s^3	6

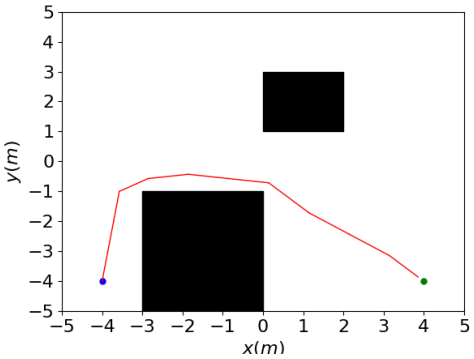

Figure 7: Velocity space

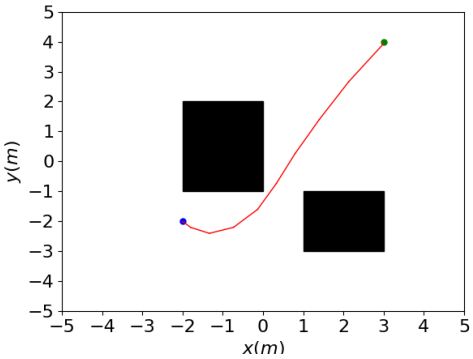

Figure 8: Acceleration space

6.2 Computational Efforts

Planning in the maps above was performed in lower dimensional spaces which took a reasonable amount of time to give the trajectories. But in the higher dimensional spaces, specifically jerk space, it is extremely cumbersome to calculate the optimal trajectory directly. In the figure below, red dots represent the states explored after 0.1 million iterations. Clearly, generation of smoother trajectory needs a better approach.

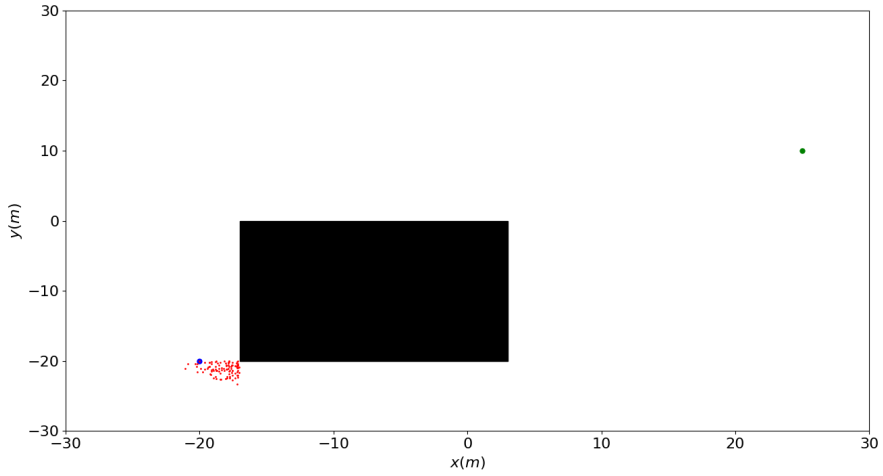

Figure 9: Exploration in jerk space after 0.1 million iterations

7 Trajectory Refinement

In the proposed planning approach, the dimension of the state space increases with increasing requirements on the continuity of the final trajectory. More precisely, if C^2 continuity is required for the final trajectory, jerk should be used as a control input and the state space of the associated second order system would be $\mathcal{R}^9$ (position, velocity acceleration). Generally, planning in higher dimensional spaces (e.g., snap input) requires more time and memory to explore and store lattices/states. In this section, we introduce a hierarchical approach to planning a feasible trajectory in high dimensional space by utilizing guidance from a trajectory planned in lower dimensional space. It is observed that the overall computation time of this hierarchical planning turns out to be shorter than the total time it takes to plan a optimal trajectory directly.

Denote the prior trajectory in lower dimensional space as Φ^p, we are searching for a trajectory in higher dimensional space $\Phi^q(q > p)$. Assume the duration of each primitive in Φ^q is τ, each lattice s_n^q in the graph is associated with a time T_n which is the minimum time it takes from the start to the current lattice. T_n is an integer multiplication of τ. Instead of calculating the heuristic $H(s_n^q)$ from the current state s_n^q to goal s_g directly, we use an intermediate goal s_n^p that is a state in Φ^p that is closest to s_n^q. Hence, the new heuristic proposed is

$$H\left(\mathbf{s}_n^q, \Phi^p\right) = H_1\left(\mathbf{s}_n^q, \mathbf{s}_n^p\right) + H_2\left(\mathbf{s}_n^p, \mathbf{s}_g\right) \tag{10}$$

where $H_1(\cdot)$ is the heuristic proposed earlier and $H_2(\cdot)$ is the cost to reach s_g from s_n^p following Φ^p.

$$H_2\left(\mathbf{s}_n^p, \mathbf{s}_g\right) = \rho\left(T^p - T_n\right) \tag{11}$$

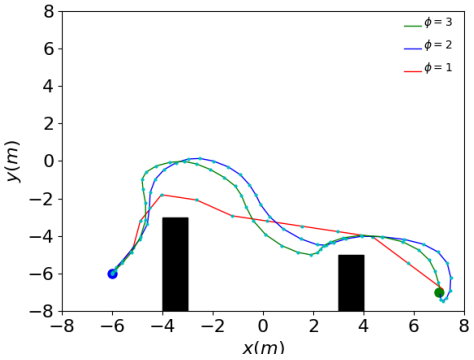

Figure 10: Map 1

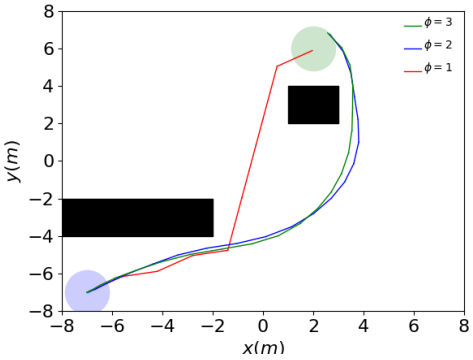

Figure 11: Map 2

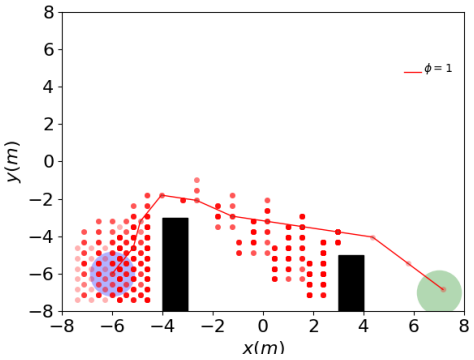

Figure 12: Velocity Space

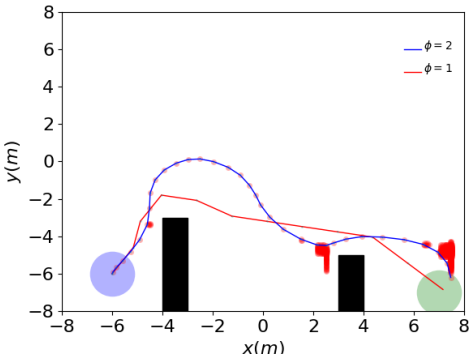

Figure 13: Acceleration space

8 Conclusions

In this article, the problem of motion planning and control of a 2D bot was addressed.

In the first phase, a cascade PID controller was designed for controlling the 2D aerial bot. Small angle approximation was used to linearize the dynamics and come up with a basic controller. This controller was not behaving well in case of constantly varying or sharply changing trajectories. Also, small angle approximation meant that the dynamics were not accurate for high values of θ. This implied severe limitations on the bot for harsh trajectories and complex dynamics such as maneuvers. To overcome this, fully nonlinear dynamic model was used. This drastically changed the outcomes and the performance was significantly better for all the trajectories including harsh trajectories as evident from the results of piece-wise linear and wavy trajectories.

In the current phase, attention was towards motion planning algorithms to generate trajectories in 2D plane. In particular, a cluttered environment with rectangular obstacles was considered. The objective was defined to be minimization of the weighted sum of flight time and control efforts used. Motion primitives were used to discretize the trajectory into piecewise continuous polynomials of time in velocity, acceleration and jerk spaces. The problem was re-formulated as graph search subject to the dynamical as well as collision constraints. A* graph search algorithm was used to find the discrete-space optimal trajectory. A distance based heuristic was introduced for the velocity space search and a modified version with distance and velocity based heuristic was used for acceleration and jerk space searches. The generated trajectories were examined with varying significant parameters such as the relative weights of time and control costs (ρ), dynamical limits, discretization, etc. For small values of ρ, the exploration took too much time before exiting any nearby obstacles. On the other hand, large values of ρ resulted in very twisted and uneven trajectories. It was observed that the graph search algorithm was computationally intractable for jerk space exploration. Hence, a new approach was introduced to refine low dimensional trajectories to gain more accuracy at the expense of sub-optimality. This approach was found

to have very small computation times compared to the optimal solution and the generated trajectories were close enough to the optimal trajectories.

References

[1] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[2] D. Brescianini and R. DAndrea, "Computationally efficient trajectory generation for fully actuated multirotor vehicles," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 555–571, June 2018.

[3] P. Foehn, D. Falanga, N. Kuppuswamy, R. Tedrake, and D. Scaramuzza, "Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload," in *Robotics: Science and Systems*, 2017.

[4] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, Oct 2017.

[5] Mellinger and D. Warren, "Trajectory generation and control for quadrotors," *Publicly Accessible Penn Dissertations*, p. 547, 2012.