Interleaved Predictive Control and Planning for an Unmanned Aerial Manipulator with On-The-Fly Rapid Replanning for Object Retrieval in Unknown Environments

by

Mohammadreza Yavari

M.Sc., University of New Brunswick, 2015
B.Sc., Isfahan University of Technology, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2022

Declaration of Committee

Name:	**Mohammadreza Yavari**
Degree:	**Doctor of Philosophy**
Title:	**Interleaved Predictive Control and Planning for an Unmanned Aerial Manipulator with On-The-Fly Rapid Replanning for Object Retrieval in Unknown Environments**
Committee:	**Chair:** Jie Liang Professor, Engineering Science
	Kamal Gupta Supervisor Professor, Engineering Science
	Mehran Mehrandezh Committee Member Sessional Instructor, Engineering Science
	Ahmad Rad Committee member Professor, Mechatronic Systems Engineering
	Mehrdad Moallem Examiner Professor, Mechatronic Systems Engineering
	Kimon P. Valavanis External Examiner Professor, Engineering and Computer Science University of Denver

Abstract

Unmanned Aerial Manipulators (UAM) are gaining attention within the unmanned aerial systems research community. They can be used for aerial manipulation tasks such as object retrieval from confined and hard-to-reach spaces. The coupled dynamics between the arm and the base and also the limited flight time would require the development and implementation of optimal motion planning and robust flight control strategies. In this work, we propose a novel integrated planning and control strategy for object retrieval. A new kinodynamic version of the RRT*, called Lazy-Steering-RRT*, is developed for planning UAM's motion from its start to a pre-grasp state, while keeping the motion of the arm to a minimum. This planning can be carried out on the fly by using Machine-Learning-based techniques to construct the edges in the search tree in a time-efficient way. This facilitates re-planning, as the environment is gradually sensed by limited range sensors onboard. Once the UAM reaches the pre-grasp state at the end of the motion cycle, an RRT approach is then utilized, where motions of the base and the arm are coordinated for reaching and grasping the object. An MPC/feedback-linearization control approach is also utilized for end-effector trajectory tracking based on a novel cross-dynamic partitioning approach between the arm and the base. The overall motion planning and control algorithms have been implemented in simulation using Multibody Physics Engines for realistic results and a number of representative simulation runs are presented. Our results show that the approach is effective in successfully executing the object retrieval task even in presence of confined spaces.

Keywords: motion planning; predictive controller; robotics

Acknowledgements

I extend my deepest appreciation and thanks to my senior supervisors, Dr. Kamal Gupta and Dr. Mehran Mehrandezh. They continuously offered sincere guidance and insightful advice throughout my PhD studies. Most importantly, they were always available if I had questions or needed help.

Also, I would like to thank my family for their endless support and encouragement in all my endeavours. I couldn't have done it without you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent years, there has been an increasing interest in using UAVs (Unmanned Aerial Vehicles) that are equipped with a manipulator, also called UAM (Unmanned Aerial Manipulators), for aerial manipulation. A UAM combines the agility and versatility of UAM with power of a robotic manipulator to address a wide variety of applications in multiple domains. For example, in Urban Search and Rescue (USAR), a UAM can provide assistance to people trapped inside collapsed buildings by delivering foods or first aid kits. It can collect samples from a construction site, or collect samples from ores. In addition, a UAM can be used for inspection in places that is hard to reach or dangerous for humans [2]; bridge inspection, high-voltage electric lines inspection, and wind turbine blade's inspection and repairs are a few examples out of many. Furthermore, UAMs can be used in production to increase the yield; harvesting fruits by UAMs is an example that is already developed [3].

1.2 Problem Definition

The aim of this research is to provide a comprehensive and practical navigation pipeline for UAMs targeting the object retrieval application. A navigation pipeline includes two major components: 1)planner 2)controller. There is generally a hierarchical relationship between the planner, that plans the trajectories over a relatively larger time horizon and the controller, which runs at much faster rate for stability and trajectory tracking. While, there are techniques such as Model Predictive Control (MPC) [4] that attempts to do both, still for non-trivial scenarios in presence of obstacles, sampling based planners such as those developed on top of *rapidly exploring random trees* (RRT) [5] provide superior performance. We opt for such a framework in this work for object retrieval with a UAM.

In the specific application of object retrieval, there are two separate phases with each phase having different requirements in terms of planning and control. These two phases of motion are depicted in figure 1.1. In the first phase of the task, the UAM is to navigate from

its initial position, shown by **A**, through the obstacles to a pre-grasp state in the vicinity [1] of the object, shown by **B** while keeping the motion of the arm to a minimum. We call this phase Folded-Arm Navigation (FAN). Once the UAM reaches the pre-grasp state, the second phase starts, where the motions of the base and the arm are coordinated to reach and grasp the object, bringing the UAM to its end target at **C**. We call this phase Full-Body Navigation (FBN). We will discuss differences in our approach to each of these phases of navigation throughout the thesis in further details.

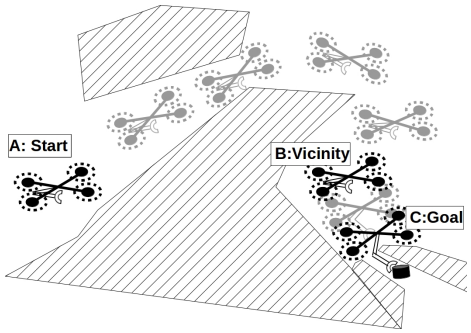


Figure 1.1: Retrieval task composed of two sub-tasks.

It is inherent to the majority of UAM missions that the environment is unknown. The onboard sensors perceive the objects on the fly and the UAM has to react to new information in real-time. This requires iterative runs of the planner, called re-planning. Since re-planning occurs online and its results are required to continue the flight, each iteration of re-planning should be very fast. This is one of the main objectives of this research to speed-up the planning component.

There are more challenges in developing a navigation pipeline for a UAM, namely, a) there are significant motion uncertainties b) energy supply is limited on a UAM, and c) UAM has a coupled nonlinear dynamics, for instance, movements of the arm shifts the center-of-mass (COM) which makes the control task difficult.

To overcome these challenges, we require a tightly-coupled planner and controller working together, iteratively. These two modules must be *fast* (for rapid re-planning) and as close as possible to *optimal* (for energy efficiency).

We present an integrated planning and control system that uses a fast *neural network aided RRT* planner* and a *partitioned controller* for a UAM for accomplishing an object retrieval [2] task in confined and hard to reach spaces. The system works in an unknown

[1] Vicinity is essentially the region where UAV could be so that the manipulator end-effector can reach the object. It will, of course, depend on the robot kinematics. For the manipulator used in this work, we approximate this region with a sphere with a 1m radius centered on the object.

[2] Object retrieval includes grasping the object in a stable fashion. However, we model the grasp as a simple attachment of the object to the end-effector, when the end-effector reaches within a distance threshold of the

environment and incorporates re-planning, i.e., sensory information gets continuously fed to the planner and accordingly the trajectory of the UAM is re-planned iteratively. We show, via simulations, that the system successfully plans and then executes UAM motions to reach the object of interest in a hard-to-reach location in several scenarios. We also simulate the limited range and field of view of onboard imaging sensors in our experiments, thereby evaluating the effectiveness of the proposed re-planning schema.

1.3 Chronological Development

During this work we first published two conference papers [6, 7] and then submitted a journal paper built on top of the conference papers with significant improvements and new contributions. We initially considered each aspect of the overall problem separately. In [7], we presented a partitioned controller that worked for a hovering UAM (UAV was hovering and not moving along a trajectory) and in [6], we presented a lazy steering neural network based kinodynamic RRT* planner for a UAV (a quadcopter with no arm attached to it), where it directly outputs the required control actions. At this stage of our research, the environment was assumed to be known a priori and the planning was carried out once in advance, i.e., there was neither any sensing nor re-planning. The main objective behind the partitioned-yet-coupled controller architecture was to address computational complexity of the design of an optimal controller for a UAM with limited on-board computation resources. On the planning side, the main objective was to design a fast planner by using neural-networks to estimate cost of the extensions within the RRT* and postponing numerical calculation of the two-point boundary value problem used for constructing edges of the solution trajectory only. We showed that the resulting planner was faster than previous RRT*-based kinodynamic planners by two orders of magnitude.

Then in the next step, we significantly improved our two earlier works and integrated them to propose a navigation pipeline, including controller and planner, for a UAM to perform *object retrieval* tasks. This renders a major distinction between our work and other works cited in the literature (See Section 1.5 for details of literature review), where they merely consider one aspect of the problem, either control or planning. We show that considering both of the problems together not only provides a comprehensive solution, but it also has additional advantages. For instance, after integration of the proposed planner and controller, we do not calculate the connections of the solution trajectory within the planning stage, instead that is left to be done at the execution time by the controller, thereby yielding 4X faster run times for the planner.

object's COM. For real implementation, a full grasp planning module along with object localization need to be implemented, which are beyond the scope of this work.

The next development towards a realistic navigation pipeline was adding a *re-planing* module. This enables the system to re-plan on a continual basis, i.e., after the first iteration of planning, based on the latest batch of sensory information. As the controller starts executing (or tracking) the planned trajectory, the planner starts the next iteration of planning, and once the next planning solution is ready, the controller executes this new trajectory starting from the closest set-point to the current state of UAM. This *re-planning* framework is critical for two reasons, i) the range and field of view of the sensors are limited and that new obstacles might be detected along the way, and ii) the UAM may deviate from planned trajectory due to uncertainties in sensor measurements and motions.

The overall motion planning and control algorithms have been implemented in simulation using Simscape Multibody [8] in Simulink for realistic results and a number of representative simulation runs are presented. MATLAB ROS [9] toolbox is used to develop communication between MATLAB/Simulink and Gazebo [10] for the purpose of visualization.

1.4 Background

1.4.1 Kinodynamic RRT*

RRT* [11] is an asymptotically optimal sampling based motion planning algorithm proposed for kinematic planning in high-dimensional configuration spaces. Kinodynamic-RRT* [12] extended this algorithm to plan for systems with differential constraints in the state space. One example which is our focus in this work is planning for UAVs during the flight. Algorithm 1 is included to review Kinodynamic-RRT* algorithm and is used to emphasize the differences with our proposed planning method. Also, figure 1.2 depicts the general structure of kinodynamic RRT*. The problem solved by kinodynamic-RRT* is to find an optimal trajectory for the system which starts in state X_i and ends in X_g. While the optimization could be with respect to an arbitrary cost function, in this research we consider the energy required for the trajectory as the cost function. The algorithm starts by creating a tree which has only one vertex, X_i. In each iteration of the algorithm a random state, X_{rand}, is generated using function $Randstate()$. Then the closest vertex (state) in the tree, $X_{nearest}$ is chosen by function $Rank()$ and the tree is extended from $X_{nearest}$ towards X_{rand} using function $Steer()$. Function $Steer()$ solves the differential dynamic equation of the system where the initial and final states are fixed, the time duration is free and system constraints such as control input limits are taken into account. This problem of finding a connection between two states of a dynamic system is the well known Two-Point Boundary Value Problem (2PBVP) [13]. The length of the trajectory given by $Steer()$ function is limited to a constant, δ. If distance to X_{rand} is more than δ, the extension ends at a different point from X_{rand}. This new point called X_{new} is added to the tree. If X_{new} and the trajectory leading to it do not collide with any obstacle and the control inputs in this trajectory satisfy

the constraints (checked in function $CollisionFree()$), i.e., the extension is acceptable, the vertex and trajectory are added to the tree. Function $Cost(X_1, X_2)$ determines the optimal energy required for the system to travel from state X_1 to X_2. The total cost of going from X_i to a vertex is sum of the costs of the edges in between; function $CostToGO()$ calculates this total cost. After finding an acceptable extension, vertices of the tree near X_{new} (given by function $Near()$) are examined to see if a lower $CostToGo$ is achievable for X_{new} (line 9 to line 13). In addition, if going through X_{new} decreases the CostToGo of a near node, the tree is modified accordingly (line 15 to line 17). Combination of the last two mentioned steps of the algorithm is called *Rewiring procedure* and is the basis of the asymptotic optimality of kinodynamic-RRT*.

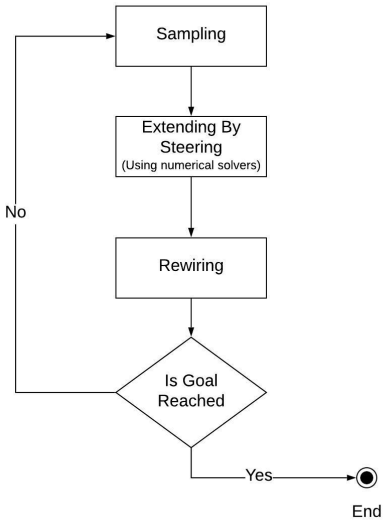


Figure 1.2: High level diagram of an optimal sampling-based planner

Algorithm 1 Kinodynamic-RRT*

1: $V \leftarrow X_i; E \leftarrow \emptyset; i \leftarrow 0$
2: **while** $i < N$ **do**
3: $X_{rand} \leftarrow RandState()$
4: $i \leftarrow i + 1$
5: $X_{nearest} \leftarrow Rank(X_{rand})$
6: $X_{new}, T_{new} \leftarrow Steer(X_{nearest}, X_{rand})$
7: **if** $(CollisionFree(X_{new}, T_{new}))$ **then**
8: $V \leftarrow V \cup X_{new}$
9: $E \leftarrow E \cup T_{new}$
10: $\boldsymbol{X}_{near} \leftarrow Near(X_{new})$
11: **for** $X_{near} \in \boldsymbol{X}_{near}$ **do**
12: $T_{near} \leftarrow Steer(X_{near}, X_{new})$
13: **if** $(CollisionFree(T_{near}))$ **then**
14: $C = CostToGo(X_{near}) + Cost(X_{near}, X_{new})$
15: **if** $C < CostToGo(X_{new})$ **then**
16: $E \leftarrow E - T_{new}$
17: $E \leftarrow E \cup T_{near}$
18: **for** $X_{near} \in \boldsymbol{X}_{near}$ **do**
19: $T_{rewire} \leftarrow Steer(X_{new}, X_{near})$
20: **if** $(CollisionFree(T_{rewire}))$ **then**
21: $C = CostToGo(X_{new}) + Cost(X_{new}, X_{near})$
22: **if** $C > CostToGo(X_{near})$ **then**
23: $E \leftarrow E - CurrentEdgeTo(X_{near})$
24: $E \leftarrow E \cup T_{rewire}$
25: **if** $GoalIsReached(X_g)$ **then**
26: *return*

Speeding up the planning algorithms is critical for high-speed applications such as navigating a quadrotor in a cluttered environment. Knowledge of the robot from the world is changed during the the travel due to movement of the current obstacles or sensing new obstacles so on-line re-planning is necessary and it must happen fast enough to avoid any collision or inefficiencies in travel of the robot. Calculating trajectories between the nodes of the searching tree and checking for collision are two known computational bottlenecks of kinodynamic RRT* algorithm shown by functions $Steer()$ and $CollisionFree()$ in algorithm 1 respectively. Implementation of functions $Steer()$ for a general nonlinear system is a challenge. For nonlinear systems such as a quadrotor there is no analytical solution presented for the problem [14]. The proposed solutions are numerical [15] which implies inherent computational burden to the algorithm. Collision checking is the other computational bottleneck of kinodynamic RRT* executed for any extension efforts which includes

checking the final state and the trajectory to be collision-free. Many of the extensions might get rejected due to collision with obstacles. In addition, if *Steer*() function doesn't check for constraints violations (on states and control inputs) internally, extensions get rejected due to constraints violations, it slows down the planning and increases the number of calls to steering and collision checking functions.

In order to avoid computational complexity of numerical solvers, there are research works [12] [16] that suggest using a closed-form solution for linear systems and extend it to non-linear systems by linearizing the system dynamic model. For nonlinear systems such as UAVs, a major disadvantage of this approach is that closed form solution doesn't consider constraints of the system such as limits on the control inputs or velocity of the system. Consequently, after finding the trajectory, it will be discarded if constraints are violated and iterations of sampling and extending must be repeated until a violation-free extension is found. This method of constraint handling requires larger number of random samples and planning iterations compared to constrained implementations of *Steer*() function. In addition, linearizing the dynamic model implies approximation which could lead to a sub-optimal or dynamically infeasible solution [15].

Authors of [17] proposed a two-stage solver to diminish the computational complexity of *Steer*() function. A purely kinematic planner is first used to find a collision-free path in the configuration space (C-space). In the second stage, this path is used as an input to an NLP solver for finding an optimal state space (S-space) trajectory that passes through the via points included in the C-space path, while taking the dynamic constraints into account. Two shortcomings of this approach are: (1) the kinematic solution may not be dynamically feasible, and (2) the kinematic solution may not belong to the same homotopy class associated with the optimal solution, hence the NLP solver may yield a sub-optimal solution only.

1.4.2 Model Predictive Control

As discussed earlier, we propose a portioned controller schema which is composed of a pair of dedicated controllers for the UAM base and the UAM arm. The UAM base has the predominant dynamics and energy consumption, therefore, the choice of controller for the base is more critical and it is reasonable to devote a larger share of the processing resources to it.

Model predictive control (MPC) is a model-based optimal approach to controlling dynamic systems. It uses numerical optimization to find control inputs that minimize both deviation from the given set-points and the control efforts. One can tune the parameters to give more weight to minimizing either of these two measures. Inefficient nature of rotary wing UAMs emphasizes the necessity of using an optimal controller and model predictive controller (MPC) is one the best choices for optimal control of UAMs, since, in addition to optimality it is also able to satisfy dynamic and kinematic constraints [18]. MPC uses

constrained optimization; meaning that we can specify linear or nonlinear constraint for the system including limits for control inputs and their rate of change and also limits for the states of the system. This is a key requirement to achieve a practical controller for a UAM. We need to determine the maximum rotational velocity of the propellers as well as the maximum pitch and roll angles. Dynamic modeling of quadcopters, which is the base of our UAM, is well studied so using a model-based controller is not a problem for us. Considering all of these aspects, we chose MPC to control the base of our UAM in this work.

The cost we pay for all the advantages of MPC is the computation power. Considering the limited resources on a UAM i.e. embedded computers, large amount of computation leads to lower control frequency which is problematic. Garimella1 et al. [19] used an unconstrained variation of model predictive controller to generate unconstrained optimal trajectory in the configuration space of a UAM based on a given Cartesian-space trajectory. However, due to the computational complexity of MPC it does not run faster than 10Hz and they could not use it as the real-time controller for the UAM.

To overcome the computation heaviness of MPC we proposed two solutions: 1) we propose the partitioned controller and use MPC for the base, as the predominant part of dynamics, only. 2) we linearize the base dynamics of the UAM base for zero roll and pitch and use linear model predictive control (LMPC) instead of MPC. Computation load of LMPC is significantly lower compared to MPC.

We provide a more detailed discussion about MPC and how we use it in this work in chapter 3.

1.5 Related Works

In this section, we discuss the related works in three categories: 1) research works on optimal control of UAMs 2) research works on rapid kinodymaic planning for UAMs 3) research works on the general application of object retrieval using a UAM.

1.5.1 UAM Controller Design

Resolving challenges related to the high degrees of freedom and the dynamic complexity of the UAMs, due to the cross-dynamic effect between the drone's base and the arm, has been the center of attention in the research community. In the work reported in [19–24] they derive a dynamic model for the whole system. The controllers used in this approach are usually not fast enough to be deployed in real systems given the limited computational resources [20] available on board. Another approach [25, 26] attempts to separate dynamic modeling and control for the UAV and the arm. In [25] they simplify the separation of the UAV and arm's dynamics by only considering the effect arm's movement will have on the UAM's COM and its inertia. Therefore, a full dynamic interaction model between two subsystems is not present. As discussed in [7], optimal controllers are computationally

expensive and separating part of the system with the dominant dynamics (UAV) from the rest of the system (manipulator) can be a big advantage. This way, one is able to design a high frequency optimal controller for the UAV which is the main energy consumer. In fact, this approach benefits from the natural partitioning between UAV and manipulator and is the one we use. However, we fully take into account the dynamic interaction between the UAV and the Arm.

[26] proposes another approach for partitioning the UAM controller where the dynamic model of the whole system is divided into linear and nonlinear parts. Linear MPC is used for the linear portion of the dynamics and the nonlinear part of the dynamic model is considered as a disturbance and is handled with a disturbance observer. The drawback of this approach is that the nonlinear part of the UAV dynamics is not controlled optimally. Separating the modeling and controllers in terms of "linear/nonlinear" instead of "UAV/manipulator" may not provide an optimal control solution for the UAV which has the dominant dynamics. Also, it does not reflect the natural partitioning (between UAV base and manipulator) of the system hence it is not generic, and is specific to the particular UAM.

In the seminal work [27], the authors provide a guideline for separating the cross-dynamic effects between system components via diagonalizing the inertia matrix for tree-like serial mechanical systems. This is achieved by converting the velocity space to new variables, while keeping the configuration space intact. This idea is applied to a UAM in [28]. A key shortcoming with this approach is that it also transforms constraints from convex to non-convex constraints after this velocity conversion, hence implementation of the optimization-based control strategies, such as that in MPC, becomes computationally more intensive, therefore, rendering it unsuitable for on-the-fly calculations using limited onboard processing resources.

Inefficient nature of rotary wing UAVs emphasizes the necessity of using an optimal controller and model predictive controller (MPC) is one the best choices for optimal control of UAMs, since, in addition to optimality it is also able to satisfy dynamic and kinematic constraints. [18, 19] used an unconstrained variation of nonlinear model predictive controller (NMPC) to generate unconstrained optimal trajectory in the configuration space of the UAM based on a given Cartesian-space trajectory. Due to the computational complexity of NMPC, in the reported work, it does not run faster than 10Hz which is too slow for control in real-world implementations. Two additional non-optimal controllers were used to perform real-time control based on the optimal outputs of the NMPC.

Lunni et al. [29] used NMPC for optimal control of a UAM consisting of a quadrotor and a 3-DOF serial link manipulator. In addition to following a trajectory by the end-effector, redundancy of the system was used to keep the center of mass close to the geometrical center of the UAM and increase the manipulability of the UAM. To achieve a higher frequency for the controller they used a simplified dynamic model by assuming zero roll and pitch angles. The mutual dynamic effects between the arm and the quadcopter is not modeled and is

assumed to be negligible or compensated by keeping COM close to the geometrical center of UAM.

In our approach, we separate the control of the UAV and the arm, while considering their cross-dynamic effects.We linearize the UAV dynamics around a hovering state and implement a model-predictive strategy to control the UAV despite arm's movements. While, it would be difficult to quantitatively and objectively compare the performance of a non-convex, nonlinear optimization-based controller with a linearized controller with convex constraints, a few conceptual advantages of our proposed method can be conjectures, which are: (1) It yields a generic and modular approach that can be readily used with any UAV and any manipulator system with different cross-dynamic effects, and (2) calculations are light, therefore, suitable for being implemented on-the-fly using UAM's onboard processing resources.

1.5.2 Kinodynamic Planning for UAMs

On the motion planning aspect, the main concern of our work is to provide a 1) *fast* and 2) *energy aware* kinodynamic planner so, in this section, we are interested in research works on kinodynamic planning that deal with these two challenges.

The major computational bottleneck of sampling-based kinodynamic planning is to find the trajectories connecting one state (node) to another, called *steering*. Steering is normally posed as a two-point boundary value problem (2PBVP) for no closed form solution for nonlinear systems is available. Instead, this is usually solved by numerical methods such as shooting methods [30]. In order to avoid computational complexity associated with these numerical solvers, some works [12, 16] suggest using a closed-form solution for linear systems and extend it to non-linear systems by linearizing the system's dynamic model. [17] proposed a two-stage solver to alleviate the computational complexity of steering. A purely kinematic planner was first used to find a collision-free path in the configuration space (C-space). In the second stage, this path was used as an input to a Nonlinear Programming (NLP) solver for finding an optimal state space (S-space) trajectory that passes through the via points in the C-space path, while taking the dynamic constraints into account.

In [31, 32] they propose the idea of using machine-learning-based estimators for finding the closest node to the new sample ($Rank()$ function), to accelerate Kinodynamic-RRT. In these works, there is no consideration about system constraints (e.g. limits on the control input and/or states in form of velocities). Moreover, steering is still implemented using numerical solvers. The time complexity of these algorithms remain at the O(N) level, where N denotes the number of random samples in the RRT. These works are limited to RRT i.e., they do not consider RRT*, i.e., the optimal variation of the RRT.

The idea of using a cost estimator to implement $Rank()$ is also proposed in [33] using a neural network. In addition, to avoid using a numerical solvers for solving the *Steer* function, a control policy is used to forward propagate from X_{near} towards X_{rand}. Forward

propagation using a control policy, however, might lead to a state different than X_{rand} in many cases. This is problematic especially in the rewiring stage of RRT*. This method adds a new node to the tree when it leads to a different state with forward propagation, instead of rewiring the current nodes i.e. a new leaf with lower cost is added instead of reducing cost of the current node and improving the current trajectories. The probabilistic optimality of this approach is proved but there is no discussion about the affect of this method on the convergence rate of kinodynamic RRT*.

The second approach for using machine learning in sampling based planners is to implement kinodynamic planning without any usage of numerical solvers [34]. In this work a single pendulum is chosen as the target system for the planning. Two machine learning methods are trained, one for estimating the *Cost* function and the other one to estimate *Steer*(). k-nearest neighbors (KNN) algorithm is used for training of the two machine learning methods. Since KNN is a "a standard non-parametric function approximator with robust performance in smaller state-spaces", according to the authors of [34], it doesn't work well for higher-dimensional systems. Furthermore, there is no constraints on the control input or the states which makes the control input estimation significantly easier. Such constraints do exist in practice and thy must be taken into account. In addition, the system used in this paper is a simple one with only one degree of freedom; we believe that this is why the control inputs are possible to be estimated. In general, it would be difficult to avoid numerical optimization completely.

Authors of [14] proposed a third approach for accelerating sampling based kinodynamic planning with offline learning. The proposed framework includes an offline phase in which *Steer*() and *Cost*() functions are solved for a set of random pairs of states and stored in a look-up table without considering any obstacles. Then, in the online phase, we have the set of connected pairs, initial and final states and the obstacles. Firstly, the initial and goal states are connected to the set of connected states by steering with a numerical solver. In this step, machine learning is used to estimate the reachable states in the connected set and reduce the efforts for connecting the initial and goal states to the set. Then FMT* [35] is used to find the optimal path using this set of samples. In contrast to RRT* which adds a new sample (state) at each iteration, FMT* works on a pre-selected set of states. It is proved that FMT* algorithm finds the optimal path that can be made by the states in the pre-selected set of states. The approach presented in [14] eliminates many numerical solver calls by creating the connected set of states offline, though it suffers from the inherent disadvantages of FMT*: i) the samples are pre-selected and fixed. In other words, in some cases a solution is not returned since the picked samples can not make a solution considering the obstacles, and ii) FMT* gives the optimal solution when the number of samples reach infinity but with limited number of samples in a high dimensional space, the solution could be far from optimality and there is no way to improve it in the online phase by taking additional steps as we do in RRT* with new samples and rewiring.

Huang et al. [36] propose a different approach for online kinodynamic planning while considering the energy limits. The purpose of this work is to provide motion plans for a UAV following a mobile target on the ground. They use forward dynamic programming to find the optimal control input to steer towards the most recent location of the mobile target. In this work problem is formulated without any constraints on the control inputs or states of the UAV. In addition, there is no obstacle in the environment. Therefore, the problem formulation is much simpler than what we address in this work.

A separate but related line of research [37] utilizes learning techniques to develop a *model-free* kinodynamic planner. The advantage of model-free approach is that no knowledge of the system's dynamic would be required, however, it assumes that the system is linear. RRT* (Kinematic) is used to find a plan from the starting point to the goal in a known environment, in an offline phase. Then, in the online phase, a reinforcement learning module finds policies to control the linear system towards the set-points on the planned trajectory. This reinforcement learning module tries to learn and move towards the optimal policy iteratively.

A limit of the above approach is that it is applicable to linear systems only, thus, can not handle the planning problem for a UAM. The second drawback is that it uses a kinematic (non-kinodynamic) planner (RRT*) to find the set-points on the trajectory. Although, the policies for moving between the set-points will converge towards an optimal solution within the kinematic framework eventually, the whole plan might be far from a kinodynamic optimal plan since it is based on a kinematic planner. In contrast, in our approach, a kinodynamic planner is proposed by means of integrating the learning based cost estimations into RRT*. Lastly, constraints on the control inputs and states of the system are not considered in the learning strategy proposed in [38], while in our approach, the constrained optimal cost is learned in the training phase and a constrained optimal controller (based on MPC) is used to make connections between the set-points during the flight.

1.5.3 Object Retrieval with UAM

[39] deals with the problem of simultaneous motion planning and control for a UAM with constraints on the end-effector's motion. In this paper a control-aware planning schema is proposed, where a controller is used to perform the steering inside RRT (not RRT*) directly. The search space is the task-space (as opposed to state-space). As mentioned in the paper, the controller is designed to follow task-space set-points not the full-state of the UAM and this approach is not applied to RRT*. [40] deals with object retrieval using a UAM as well. Authors propose a locally-optimal trajectory planner for a UAM, however it is assumed that the dynamics of the arm is negligible thereby dynamics of the UAM is simplified and it becomes a *flat system*. The flatness property is used to define the steering as a sequential quadratic programming (SQP) problem that can be solved almost in real-time. Also it is

assumed that the environment description is given by a mesh of convex polygons thereby collision avoidance can be carried out at high rate. [41] proposed a very similar approach for UAM object retrieval. They model the UAV base in X-Z plane (a sagital plane) only using X, Z and pitch angle as configuration parameters. They show that this model is flat and they use this feature to calculate the trajectory of the UAM under a quadratic programming framework.

In our work, we propose a general approach i.e. it applies to any type of arm attached to the UAV base and that it can adapt to any feasible payload that it would carry. We do not assume negligible dynamic interactions between the UAV base and the arm, hence we treat it as a non-flat system. Also, our method works under on-the-fly reactive operation constraints required when incrementally sensing the environment as opposed to that assuming a full a priori knowledge of the world.

A key distinction between our approach and works like [39] is that our proposed planner is online and replanning is a key aspect of our system. In [39] planning for UAM is designed as an offline procedure which plans a trajectory in advance and then the controller is to follow the set-points on the fly. There are two main drawbacks in this approach. The first is that it requires a complete a priori map of the *world*, before the flight starts, to be used for planning. This is not the case found in real applications. Instead, a UAM is often equipped with some ranging sensors like Lidar or RGBD cameras and it can *observe* a portion of the world around its current state. As a result, the initial understanding of the world is not complete and it changes as the UAM flies within the environment and collects information about it incrementally using its on board sensors. The second drawback is that with offline planning, the controller is required to follow the planned trajectory all the way to the goal. In reality, due to different types of uncertainties, such as modeling errors, localization errors, etc. UAM might deviate from the planned trajectory. In this work, we propose a continual iterative re-planning and control schema to address both of these issues. This schema is inspired by our earlier work on re-planning [42], as well as some other works [43] in the context of ground based mobile robots.

A separate, but complementary branch of works on aerial object retrieval is on vision-based detection and grasping of the objects [44]. A key focus of this body of work is on object detection and localization whereas in our research, the focus is on solving the planning and control problem in presence of obstacles in cluttered environments.

1.6 Contributions

The main contributions of this work are summarized as follows:

- We propose a *complete yet partitioned* modelling of UAMs which reflects the natural partitioning between the UAV and the arm and makes it possible to use a decoupled and modular control scheme with an optimal controller devoted to the dominant

dynamics (i.e. dynamics of UAV). A key aspect of this work is a *"disturbance rejection"* scheme which compensates for mutual effects of the UAV and manipulator on each other. The disturbance rejection scheme uses the manipulator's dynamic model and state to accomplish this task.

- Using neural networks, we propose a new variation of kinodynamic RRT* - Lazy Steering RRT* - which expands the search-tree without calculating the trajectories in the expansion phase. The actual trajectories and the corresponding collision status of the them are computed once a potentially-optimal trajectory has been determined. This lazy steering approach is shown to be two orders of magnitude faster than some of the recent approaches.

- We target the object retrieval application with a UAM and take advantage of the natural phasing of the task to provide an interleaved planning and control pipeline to achieve the task. Two different planners (one kinodynamic and the other kinematic) and two different modes for the controller (folded- and moving-arm) are used efficiently during these phases.

  - We strive to achieve a balanced compromise between optimality and run time in different sub-tasks and propose a coupled pair of controller and planner, each of which are optimal of their own right, albeit modified towards achieving fast run times.
  - We consider realistic scenarios in which the environment is unknown and proposed a re-planning schema.

Partial results from this thesis have been published in conference articles [6, 7, 45] and a conditionally accepted journal article [46].

Chapter 2

Lazy Steering RRT* Algorithm

In this chapter we propose a novel planning approach for UAVs. The main emphasis is to provide a practical planner by avoiding assumptions such as a priori knowledge about the environment or following the trajectories with no deviation. We address these real-world uncertainties by proposing a rapid planner and continues replanning. In other words, we do not expect the planner to provide a solution that is followed perfectly all the way to the goal; instead, we want to have a good plan for the next few iterations of the controller and then we have a new plan based on the new understanding about the world and the recent state of the UAV.

We start off the chapter with how our re-planning system is structured and then present details of the core planner. At the end of the chapter we present the results of evaluating the proposed planner in simulation.

2.1 Importance of Rapid Planning

In this chapter we present a novel approach to kinodynamic planning which is faster than the previous methods. As a result, we are able to run the proposed planner repeatedly, i.e., replan on the fly and provide the controller with a new plan. In some works, such as [39], planning for UAM is designed as an offline procedure which plans a trajectory in advance and then the controller is to follow the set-point on the fly. There are two main drawbacks in this approach. The first is that it requires a complete a priori map of the *world* before the flight to be used for planning. This is not the case in real applications. Instead a UAM is often equipped with some ranging sensors like Lidar or RGBD camera and it can *observe* a portion of the world around its current state. As a result, the initial understanding of the world is not complete and it changes as the UAM flies. The second drawback is that with offline planning, the controller is required to follow the planned trajectory all the way to the goal. In reality, due to different types of uncertainties, such as modeling errors, localization errors, etc. UAM might deviate from the planned trajectory. In this work, we propose a continual iterative planning and control schema to address both of these issues.

This schema is inspired by our earlier work on re-planning in the context of ground based mobile robots [42], as well as some other works such as [43]. At each iteration of the planning, world's map is enhanced by means of adding the newly sensed information. The planner uses the latest version of the map, so any newly sensed obstacles will be avoided by the next planned trajectory. Also, if for any reason the UAM deviates from the planned trajectory, the new plan starts from the current state of the UAM and mitigates the problem.

Figure 2.1 depicts the state-machine description of the planner module along with the *set-point manager* module. These two modules run simultaneously in two different threads. The planner module plans iteratively from the current state of the UAM to the goal and publishes the planned trajectory. This repeats until the goal is reached. The set-point manager module, reads the published trajectory, and sends the next state of the *current* trajectory, the set-point, to the controller and waits until the set-point is reached by the controller. Before sending a new set-point, it checks if a new trajectory has been planned by the planner. If so, it finds the state in the newly planned trajectory that is closest to the current state and sets it as the next set-point and the process repeats.

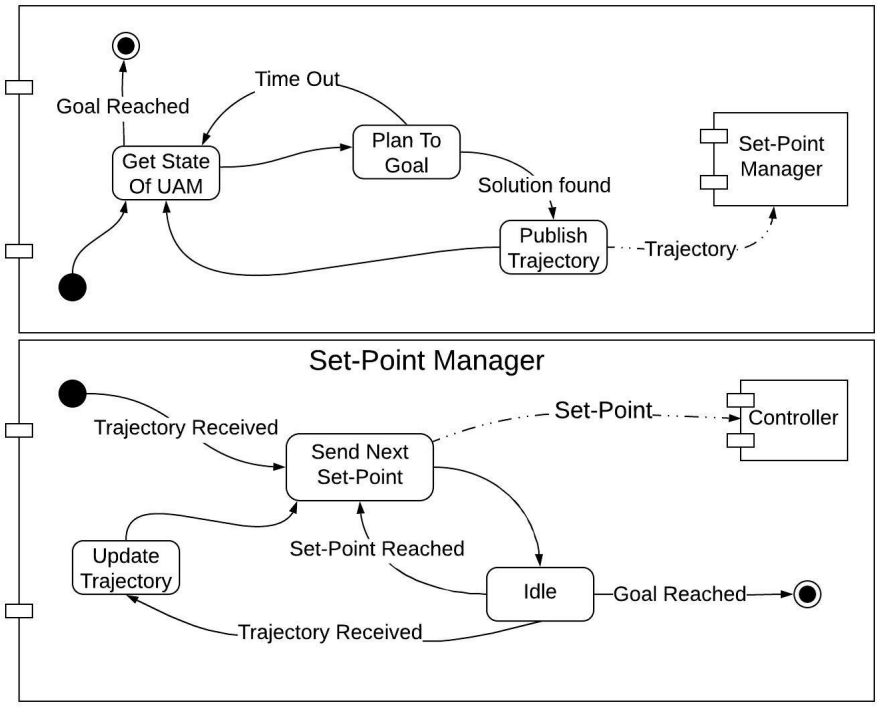


Figure 2.1: Simultaneous planning and set-point tracking. The set-point manager sends the next set-point to the controller as its next goal.

Figure 2.2 depicts an example of this scenario. Plan I is planned at the time that only the red obstacles is within the range of the sensors. As a result, it does not avoid the yellow obstacle going towards the goal. After some iterations when the UAM is at point Q, the

sensors detect the yellow obstacles therefore the planner outputs a trajectory that avoids the yellow obstacle as well, called plan II (shown in green dashed line). The purple dotted line shows the executed trajectory of the UAM as the result of following plan I first and then plan II.

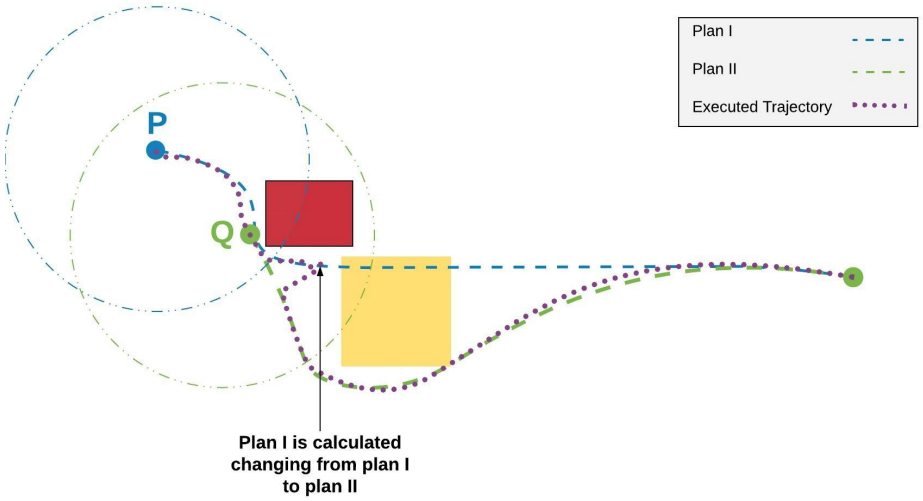


Figure 2.2: Real-time planning with limited FOV of sensors. The trajectory is re-planned as world's map is being developed

As can be seen in figure 2.2, the UAM follows plan I for some iterations even after start of the plan II. That is the computation time required for one iteration of the planner, Δ_c. This timing relation is explicitly and clearly shown in figure 2.3 which shows the timeline of the proposed iterative re-planning. After a solution plan is ready, it starts to be executed and at the same time a new planning session is started from the current point. Once the new plan is available (after time Δ_c), the new plan starts to be executed.

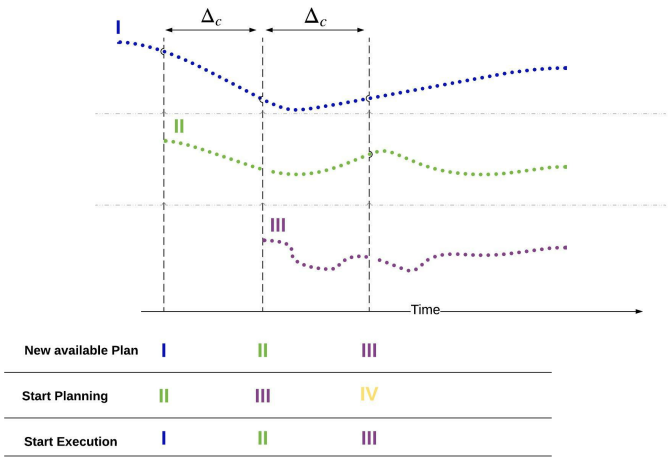


Figure 2.3: Time line of iterative planning and execution. The last plan is used for execution until the next plan is ready.

2.2 Lazy Steering based on Machine Learning

As mentioned earlier in section 1.4, a key computational bottleneck in a kinodynamic RRT* planner is the non-linear (in general) solver for connecting two nodes of the tree, often called the steering algorithm. To achieve a fast planner, we propose the idea of lazy steering. We call it lazy steering since the steering part of planning is postponed to a latter stage. We address this computational bottleneck by using neural networks (NN) to estimate the cost of the trajectory (without actually calculating the trajectory) instead of using a non-linear solver for steering. We call this cost estimation neural network C-NN. A second neural network, called F-NN, is trained to identify the feasibility of a connection. F-NN is trained to return 1 if the constrained two-point value problem define by the two vertices are solvable and 0 otherwise. We enforce system dynamics and control input velocity constraint in generating the training data. For example, we have maximums for roll and pitch angles and the linear velocity of the UAM as well as the maximum force the propellers can generate. As a result, at planning time, F-NN helps to avoid trajectories that are not achievable for execution due to constraints or limits of the numerical solver. We discuss the details about F-NN and C-NN in the sections 2.2.1 and 2.2.2 respectively.

Figure 2.4 depicts the flow of going from a standard kinodynamic RRT* (based on numerical solvers) to LS-RTT*, a fast NN-based kinodynamic RRT*. In LS-RRT*, as depicted in Figure 2.4, the edges are not connected but the tree is expanded just by estimating costs of the connections.

Figure 2.5 shows how a NN estimator fits into a kinodynamic RRT*. The main difference between the original kinodynamic RRT* and our NN-based variation is that in the latter, search tree is growing without calculating edges, i.e., just by adding nodes and the

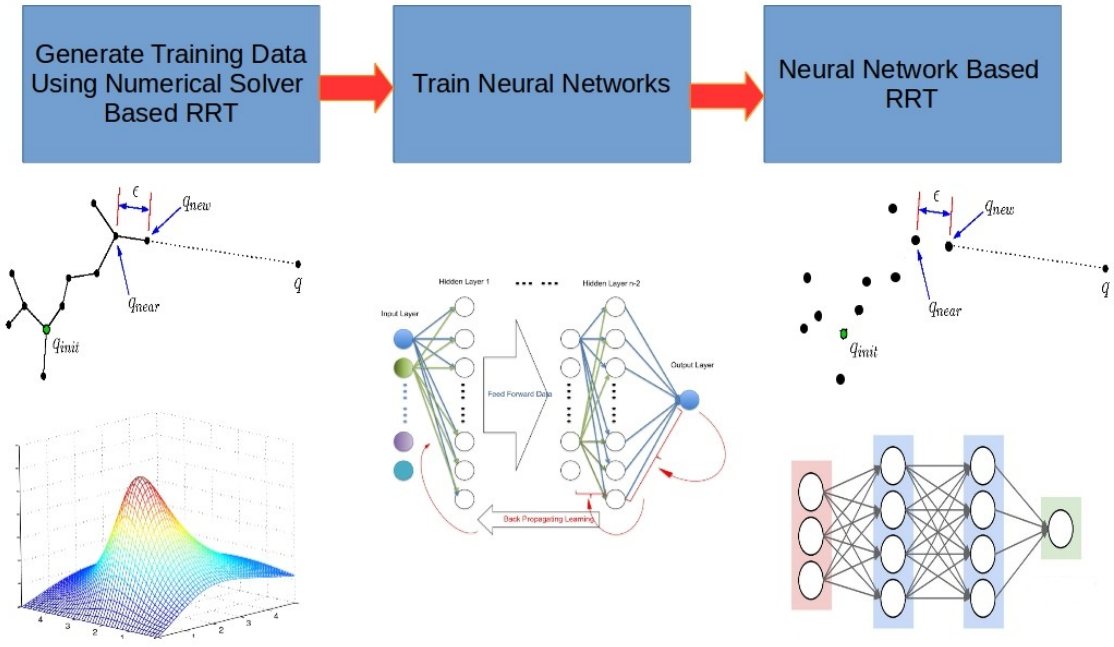


Figure 2.4: Developing NN-based RRT* from the original RRT*. Please see text for explanation.

estimated cost of the trajectory connecting a node. In [6] we proposed that after finding an optimal solution, connections between the set-points (two point boundary value problems) are calculated using a numerical method (multiple-shooting method) to find the control inputs that are required for going from one node to the next one along the solution path. In addition, solving the connections confirms if there are any collisions between the connections and the obstacles. In this work, we have a controller to decide on the control inputs needed to follow the set-points, therefore, we don't need connections to be solved by the planner. Instead we pass the set-points along the solution path directly to the controller. This modification eliminates about 50% of the CPU time of the planner. Regarding the collision checking, vertices are checked for collision during the planning. In the rare case that the planned trajectory leads to collisions, the controller stops the UAM from colliding and the planner starts a new iteration of planning with a reduced step size, i.e. vertices are closer to each other. We added an inflation layer to the obstacles to make sure that the controller has enough time to stop the UAM before any collision.

As stated in the introduction, LS-RRT* is used for FAN phase of the task, i.e., from point **A** to point **B** in figure 1.1, in which UAM has to go over a large distance through the

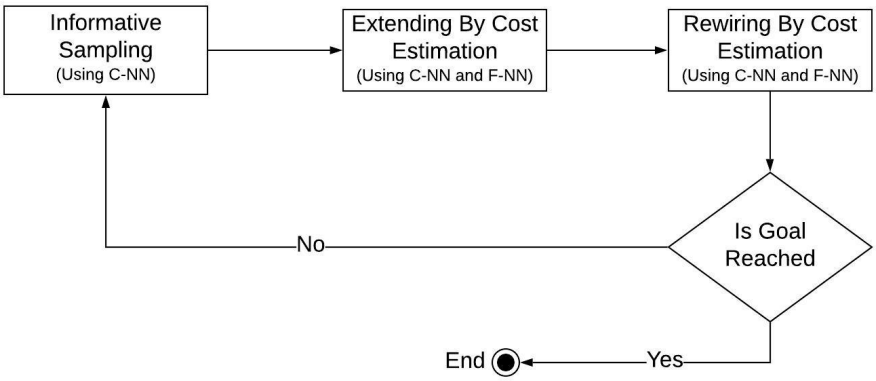


Figure 2.5: Using neural network estimators to speed up kinodynamic planning

obstacles towards the object of interest while the manipulator doesn't need to move and the simpler dynamic model helps us train C-NN with higher accuracy.

2.2.1 F-NN: a Neural Network to Classify Feasible Extensions

F-NN is used to identify *feasible* extensions. We call an extension feasible if it is solvable by the NLP solver and does not have large deviations from the direct line connecting the two states. Figure 2.6 depicts a large deviation connection in three dimensional space. The two green spheres are the states of the connection, the curve shown by the brown spheres is the solution for this extension given by NLP and the purple cylinder shows the acceptable deviation from the direct line between the two states. In section 2.3 we discuss the advantages of discarding high-deviation extensions as well as its costs.

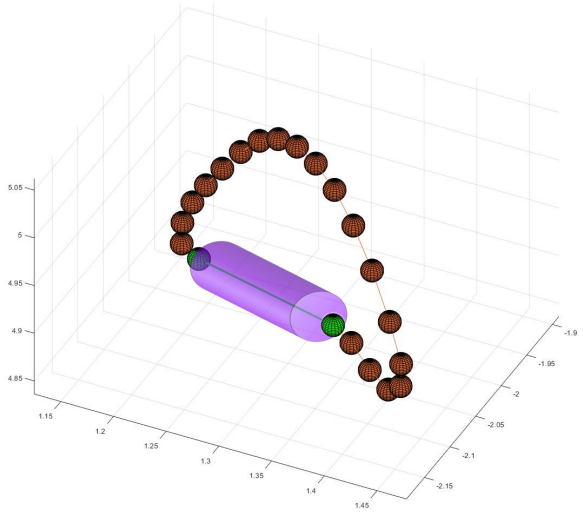


Figure 2.6: F-NN discards an extension if the solution is expected to go out of the cylinder.

F-NN is a fully connected neural-network with 4 layers with $\{10, 30, 10, 1\}$ neurons in respective layers. Significant amount of experiments is spent to determine the hyperparameters of the network including the number of layers, neurons at each layer and activation functions. Hyperbolic tangent sigmoid activation function is used for the hidden layers of F-NN and linear activation for the output layer. Input of F-NN is a vector of size 24 which is composed of the two states which we want to connect to each other. Output is a value in $[0, 1]$; where values greater than 0.5 indicate a feasible extension.

The training data is generated by running kinodynamic-RRT* with a numerical NLP solver as the steering function. For this purpose we use CasADi library [47] for the NLP solver which uses the interior point optimization method [48]. We started with $200,000$ of pairs of states chosen randomly and increased the size of the data-set in a few steps up to $400,000$ pairs that resulted in an acceptable accuracy. The training data-set is divided into training (80%), validation(10%) and test(10%) data-sets which is a standard approach for training neural networks. These data-sets are used to train both F-NN and C-NN.

F-NN provides over 90% of accuracy, which means that false-positive and false-negative errors sum up to 10% in total. Errors in F-NN prediction are detected at the end of the

planning once the solution is solved by the NLP solver and the trajectories are checked for collisions. If an extension is not solvable or results in collision we discard it and repair the solution by more search until the two sections of the search tree are connected.

2.2.2 C-NN: a Neural Network to Estimate Cost of Extensions

The second neural network, C-NN, is used to estimate the energy-cost of connecting two states without calculating the trajectory using NLP solver or simillar approaches.

C-NN is a fully connected neural-network with 3 layers and and $\{30, 50, 1\}$ neurons respectively. The activation function is hyperbolic tangent sigmoid for the hidden layers and Log-sigmoid for the output layer.

Input vector of C-NN is the same as for F-NN, concatenation of the two states that are to be connected. Output of C-NN is a real value which represents the estimate of the optimal constrained cost of the extension if it would be solved by the NLP solver.

In this work we define the energy-cost of the UAV as follows:

$$\kappa(t) = \omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2 \tag{2.1}$$

where ω_i is the rotational speed of the ith propeller of the UAV at time t. So we want to minimize

$$\int_{t=0}^{T} \kappa(t) \tag{2.2}$$

where T is total time duration of the trajectory.

Training data is generated identically for F-NN and C-NN. The energy-cost of the solutions found by the constrained NLP solver is used as the expected output of C-NN.

C-NN estimates the test data-set with accuracy of 98.06%. The normal distribution fitted to the error percentage of cost estimation of more than $400,000$ extensions is depicted in figure 2.7. For this normal distribution the average error is 1.86 and its standard deviation is 3.39.

2.3 Lazy Steering RRT* Algorithm

Algorithm 2 describes how neural networks are used in lazy-steering RRT* to provide an efficient constrained optimal kinodynamic planning. The underlined functions are the differences between our approach and the original method summarized in algorithm 1 in section 1.4. Rest of this section is dedicated to explaining functions used in this algorithm.

Rank

$Rank()$ function is used to find $X_{nearest}$ the closest vertex of the tree to X_{rand}. In most of the works the distance metric used in ranking is the same as the distance metric used in rewiring of RRT*. In this work, however, we use 12-dimensional euclidean distance for

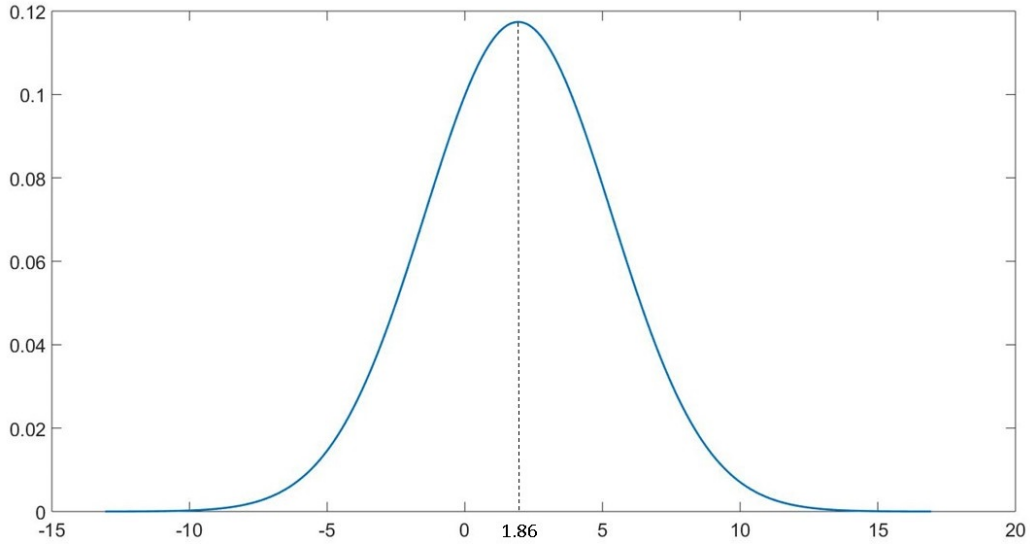


Figure 2.7: Normal probability distribution for estimation error of the cost function.

ranking to explore the state space quickly and find a way around the obstacles while cost estimations by C-NN are used for rewiring to provide the optimal solution in terms of the energy cost. In the simulations, we found that using the euclidean distance leads to faster run-time for finding the initial solution. This topic is discussed in [31] as well. They compared performance of RRT (not RRT*) when euclidean distance is used and when the learned cost estimator is used for ranking. Their results show euclidean distance ranking leads to finding a solution in significantly shorter time (by a factor of more than 18) but at the cost of loosing path quality in terms of smoothness. In this work, we rely on rewiring procedure, which uses the cost (energy) estimator, to improve the path quality.

PseudoSteer

In our method, there is not conventional steering to find a trajectory between two states. This is postponed to the end of algorithm where trajectories involved in the final solution are calculated. Instead, function $PseudoSteer(X_{nearest}, X_{rand})$ returns X_{new} which is the new node of the search tree. In other words, the search tree of lazy-steering RRT* has no edges up to the end of planning and it is composed of nodes only. As a result, the set of edges, called E in the original algorithm 1, doesn't appear in our algorithm and the structure of the search tree is held by saving the a parent node for each of the vertices. In order to find X_{new}, we use the same steering function used in kinematic RRT* [11] which is based on euclidean distance.

Algorithm 2 Kinodynamic-RRT*

1: $V \leftarrow X_i; i \leftarrow 0$
2: **while** $i < N$ **do**
3: $X_{rand} \leftarrow RandState()$
4: $i \leftarrow i + 1$
5: $X_{nearest} \leftarrow \underline{Rank}(X_{rand})$
6: $X_{new} \leftarrow PseudoSteer(X_{nearest}, X_{rand})$
7: **if** $(\underline{IsValidExtension}(X_{nearest}, X_{new}))$ **then**
8: $V \leftarrow V \cup X_{new}$
9: $\boldsymbol{X_{near}} \leftarrow \underline{Near}(T, X_{new})$
10: $X_{min} \leftarrow X_{nearest}$
11: **for** $X_{near} \in \boldsymbol{X_{near}}$ **do**
12: **if** $\underline{CanImprove}(X_{new}, X_{near})$ **then**
13: $X_{min} \leftarrow X_{near}$
14: $X_{new}.parent = Xmin$
15: **for** $X_{near} \in \boldsymbol{X_{near}} \setminus X_{min}$ **do**
16: **if** $\underline{CanImprove}(X_{near}, X_{new})$ **then**
17: $\underline{X_{near}.parent} = X_{new}$
18: **if** $GoalIsReached(X_g)$ **then**
19: $FullPath \leftarrow \underline{SolveConnections}(path)$
20: **if** $CollisionFree(FullPath) == False$ **then**
21: $FullPath \leftarrow \underline{Repair}(FullPath)$
22: $Return$

IsValidExtension

Function $IsValidExtension()$ at line 7 is composed of three checks as shown in algorithm 3:

- Is X_{new} collision free?
- Is the edge between $X_{nearest}$ and X_{new} feasible and low-deviation?
- Is X_{new} a promising sample?

Algorithm 3 IsValidExtension Function

1: **procedure** IsValidExtension($X_{nearest}, X_{new}$)
2: **if** $CollisionFreeState(X_{new}) == False$ **then**
3: $Return\ False$
4: **if** $F\text{-}NN(X_{nearest}, X_{new}) == False$ **then**
5: $Return\ False$
6: **if** $IsPromising(X_new) == False$ **then**
7: $Return\ False$
8: $Return\ True$

collision checking

The new vertex is checked if it is in collision with obstacles or not. As discussed earlier, collision checking of the edges is only done at the end of the algorithm and only for the edges that contribute to the optimal solution, line 20 of algorithm 2. This lazy collision

checking [49] is originally proposed for kinematic planning and assuming all the edges to be a line segment. In the case of kinodynamic planning, edges could have any shape though. As discussed below we use *F-NN* to limit shape of the extensions and make lazy-collision checking effective for kinodynamic planning.

connection feasibility

F-NN is used to discard large deviation and non-feasible extensions. Extensions with large deviation from the direct line connecting the two states have significantly higher chance of collision. One can argue that a longer trajectory has more points in the space so the chance of collision is higher. In addition, we confirmed this empirically by running simulations. If we include this type of connections in the solution, it is very likely to find a collision once the edges are checked for. In most of the cases, a large-deviation extension is result of disagreement between velocities chosen and the direction of movement on C-Space and consumes more energy compared to a low-deviation. In the case all low-deviation extensions are blocked by obstacles or system constraints and the large deviation connection is a part of the optimal planning solution, RRT* will converge to that connection by adding samples on the curve and dividing it to a series of low deviation extensions.

promising state

After finding the first path from X_i to X_g, we are looking for samples (states) which can improve the current solution. In [50] informed RRT* is introduced. In this variation of RRT*, after finding a path, states which may not create a solution with lower cost than the current solution are not sampled. Since this method is about kinematic planning line segment connections and euclidean distance is used to determine the minimum path from X_i to X_g which passes through random state. In this case it is possible to formulate the subspace of *promising* states and sample from this subspace directly. In kinodynamic planning energy-based cost function is used instead of the euclidean distance so the method used in [50] does not work. It is an open problem to formulate the subspace of promising states considering dynamic distance. In this work we use a rejection schema to extend the informative sampling idea to kinodynamic planning, for the first time to the best of our knowledge. After a new state, X_{new}, is determined we calculate the minimum cost from X_i to X_g which passes through X_{new} using C-NN. If this minimum cost is more than the current solution cost, we discard X_{new} and start a new iteration of Kinodynamic-RRT*. Our simulations show an average speed-up of 5% when we use this method in planning.

Near

This function selects which nodes of the search tree participate in rewiring procedure. We use euclidean distance in configuration space and choose nodes within a limit from X_{new}

to be checked for rewiring. The distance is limited in C-space to keep size of the steps in tree small which helps passing through narrow passages. In addition, we want to avoid large steps as we don't check edges for the collision till end of the planning and large edges are more prone to have collisions. We do not enforce a limit on velocity difference between X_{new} and *near* nodes as it leads to larger cost improvements in rewiring while doesn't harm with narrow passage exploration nor collision likelihood.

CanImprove

Function $CanImprove(X_m, X_n)$, described in algorithm 4, returns true if choosing X_n as parent of X_m reduces cost of traveling to X_m from the tree root. In our algorithm, function $CanImprove()$ replaces lines 14, 15, 21 and 22 of algorithm 1 for rewiring. Costs of the connections in the tree are estimated using C-NN and are the base of comparison in function $CanImprove()$.

Algorithm 4 Decide if rewiring improves

```
1: procedure CanImprove(X_m, X_n)
2:     P_current ← Path(X_i, X_m)
3:     P_new ← Path(X_i, X_n) + X_n, X_m
4:     if PathCost(P_new) < PathCost(P_current)
   then
5:         Return True
6:     else
7:         Return False
   =0
```

Cost

In kinodynamic planning, cost of the planned path must reflect the energy required for the system to travel the path. We use C-NN to estimate the cost function in equation 2.2 for each extension.

SolveConnections

Before calling this function, the search tree that we generate have pseudo-edges only i.e. we know the optimal via-points for traveling from X_i to X_g, but we don't have the trajectories between the via-points. $SolveConnections()$ function uses numerical NLP solvers to calculate the connections between the via-point in the found solution. This is the main contribution of this work to limit usage of the numerical solvers to steering for connections that are in the final solution.

Repair

After solving the steering for the edges of the solution, edges are checked for collisions. In the case of a collision, we *repair* the tree. This repairing process is described in [49] in

details. In summary, the colliding edge is removed from the tree and searching continues to connect the two sections of the tree. The lazy-collision checking alleviates one of the main computational bottlenecks of sampling-based planning by limiting collision checking to the edges of the solution. However, if number of collisions in the solution is large, then many efforts are required to repair the solution. As mentioned earlier, we use F-NN to reduce chance of collisions and avoid this problem.

2.4 Results

We implemented Lazy-Steering-RRT* in MATLAB and ran simulations using an Intel Core i7 CPU. The multiple-shooting method is implemented using CasADi library [51] to solve NLP of steering.

A quadcopter, a nonlinear under-actuated system with redundancies, was used for case study. We implemented the NLP solver using the dynamic model of a typical quadcopter and trained our neural networks for this system. Quadcopter dynamics are well studied in literature [52], so we are not covering them here.

Figure 2.8 shows an example result of planning with LS-RRT* projected on to X-Y-Z space. The red spheres are obstacles in the environments and were placed randomly. Sub-figure on top-left depicts obstacles, initial and goal states and search tree at 1000 nodes. On top-right the complete search tree is depicted after a solution is found. It is observable that the tree is growing towards the unexplored regions which is an indication of a normal RRT-based planning algorithm. Sub-figure on bottom-left depicts the solution found by LS-RRT*. Lastly, on bottom right there is a zoomed-in version of the solution which shows the NLP solutions in green and the vertices in purple. Also, Euler angles of the solution trajectory are depicted in figure 2.9.

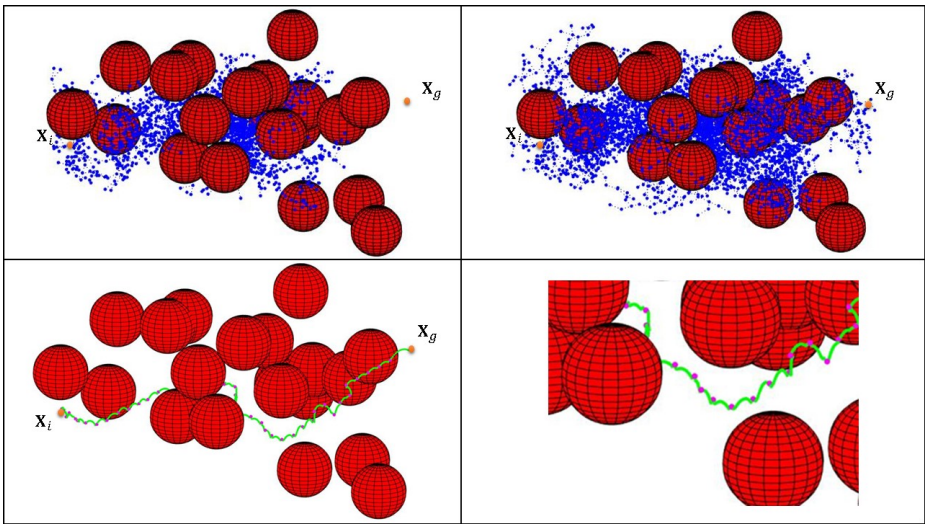


Figure 2.8: Planning results for lazy steering RRT*. Please see text for explanation.

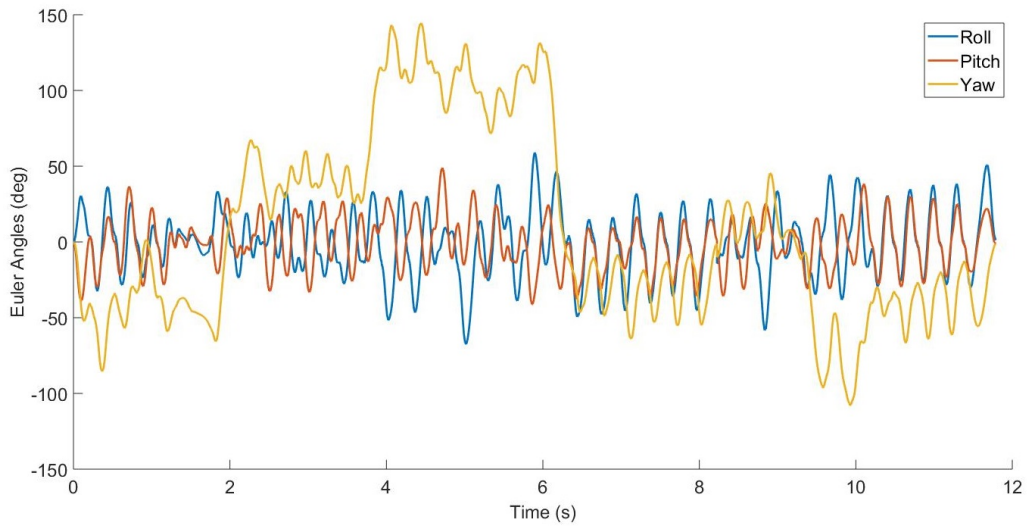


Figure 2.9: Euler angles of the quadcopter.

Control inputs corresponding to the solution of this example are depicted in figure 2.10. w_i is the rotational speed of the ith rotor of the quadcopter. The lower and upper limits on each of the control inputs, i.e., the rotational speed, are 0 and 3500 rpm, respectively. Figure 2.10 shows that the planning algorithm has been successful in satisfying these control limits. This result implies successful design and implementation of F-NN since the trajectories are not explicitly solved up until the solution path is determined, so constraints could not be checked explicitly during the expansion of the tree.

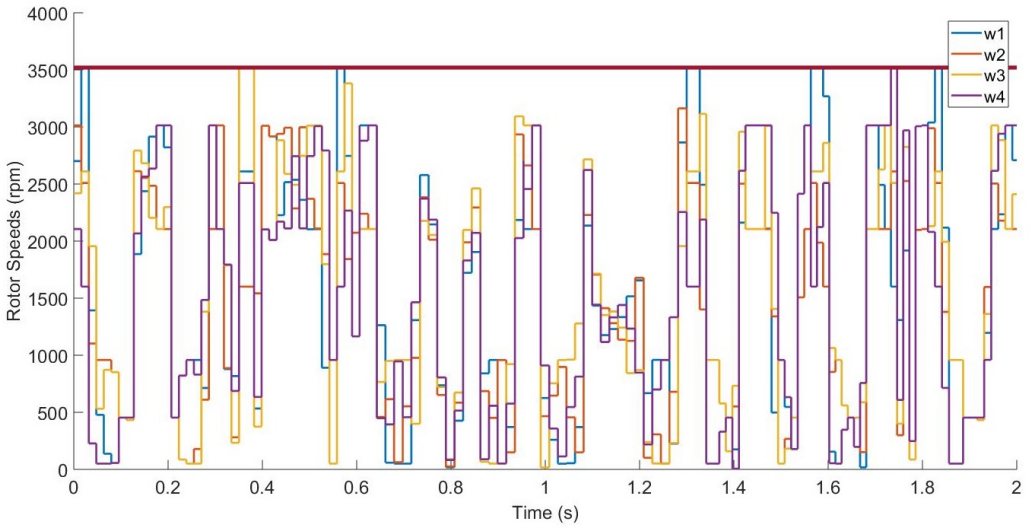


Figure 2.10: Control inputs for the first 2 seconds of the trajectory. Rotors speed are limited within the specified envelope of constraints.

Table 2.1: LS-RRT* vs NLP-Extensions-RRT*

Scenario	LS-RRT*		EasyRanking-RRT*			
	Average CPU Time (s)	Average NLP Calls	Average CPU Time (s)	Average NLP Calls	Cost (Base is EasyRanking-RRT*)	Speed-Up
1	24	43	4636	9184	1.02	190X
2	20	46	4388	9010	1.01	210X
3	29	27	5908	11189	1.04	200X
4	17	29	5541	11232	1.01	330X
5	32	39	5779	11281	1.00	180X

In order to evaluate the speed-up factor, we also implemented a planner that does the ranking without solving 2PBVPs similar to what was proposed in [31] and [32]. In this implementation NLP solver is used for creating the extensions and for rewiring. We call this method "EasyRanking-RRT*". We chose EasyRanking-RRT* for comparison since it is applicable to a complex system such as a UAM and is the most related work to the proposed LS-RRT* in terms of using machine learning to eliminate computations in Kinodynamic-RRT*. The results, as we mention below, show the significance of eliminating steering computations in addition to those for ranking.

We ran the simulations in 5 different scenes with different number of obstacles and different initial and goal states to examine sensitivity of our method to different environments. The results shown in Table 2.1 are averages from several runs for each scenario. *Extra-Energy %* compares cost of the two solutions and shows how much we gain in terms of energy saving using machine learning estimators instead of a numerical solver. *Speed-up* is the ratio between CPU time of LS-RRT* and NLP-Extensions-RRT*. The results indicate that our approach accelerates the planning by two to two orders of magnitude at the cost of loosing only up to 4 percent of energy-optimality. Also, it can be inferred that out approach is advantageous for all types of environments either cluttered or without obstacles.

It is observable that the CPU time of each run is mainly proportional to the number of NLP calls. In LS-RRT*, number of calls to NLP solver is equal to the number of edges in the final solution while number of NLP calls for EasyRanking-RRT* increases with the grow of the search tree to execute steering and rewiring. In terms of big picture complexity, LS-RRT* is of $\mathcal{O}(m)$ where m is number of the states in the final solution whereas EasyRanking-RRT* is of $\mathcal{O}(n+l)$ where n and l are the number of samples and the number of rewiring instances throughout the planning, respectively.

2.5 Summary

In this chapter we proposed a Lazy-Steering Kinodynamic RRT* that results in a speed-up of at least two orders of magnitude as compared to recent works. In our proposed method, the search tree grows by neural-network-based estimations by: i) finding out if a trajectory is feasible between two states, and ii) calculating the cost-to-go, both being achieved without explicitly solving for the trajectory thereby avoiding computationally-expensive numerical solvers. Once a potentially- optimal solution is found, the trajectories between the nodes are explicitly calculated. We used the errors in neural-network- based cost estimates to provide confidence levels in the optimality of the proposed solution. We implemented the Lazy Steering RRT* for a quadcopter with nonlinear dynamics and actuator limits. Simulation results show a speed-up of at least two orders of magnitude over some recently reported works with a slight increase in the energy cost (less than 4%)

Chapter 3

Partitioned yet Complete Modeling and Control of a UAM - Case Studies on Rotary Wing UAVs with Fixed and Tilting Rotors

A UAM is composed of two modules 1) a UAV base (a quadcopter in our case) 2) a manipulator attached to the base. One approach to the dynamic modeling of this system is to drive equations for the UAM as a single module. This approach is well studied in the literature [19–21]. As mentioned in the introduction, however, we have a different approach. We model the quadcopter and the manipulator separately and then make sure that all the dynamic cross-effects of the two module are taken into account. We call this approach to modeling a UAM *partitioned-yet-complete* modeling and discuss its advantages in details in the following subsections. The driven dynamic models are then used for model-based control of the UAV and the manipulator as discussed later in this chapter.

There are two advantages to this partitioning approach: first it allows us to develop a high-frequency optimal controller (a linear MPC) solely for the quadcopter, which has the dominant dynamics. Without partitioning, there is not a single valid point for linearizing the nonlinear dynamic model of the whole UAM while linearizing at hovering point gives a very good estimate of the separate dynamics of the quadcopter; this allows us to use a much faster linear MPC instead of non-linear MPC. Secondly, this partitioning makes our system completely modular in the sense we can change each of quadcopter and arm including the respective controllers independently and our scheme is still applies. This modularity gives a tremendous advantage to our scheme and makes it very general applying to different UAV and Arm hardware as well as their respective controllers.

In the rest of this chapter we will discuss dynamics of our UAM and then we propose controllers for the UAV and the manipulator and discuss how the cross-effects are managed in our approach.

3.1 UAM Kinematics

In this thesis our UAM is composed of a quadcopter and a two link arm as shown in figure 3.1. O_i is the inertial frame and O_b is the body frame. The transformation between O_i and O_b is described by $[\eta, \xi]$, where $\eta = [\phi\ \theta\ \psi]^T$ is vector of Euler angles of the quadcopter and $\xi = [x\ y\ z]^T$ is vector of position of the quadcopter. The manipulator kinematics are defined by $\Theta = [\theta_{j1}, \theta_{j2}]^T$ as a regular two link serial manipulator [53] mounted on the origin of O_b.

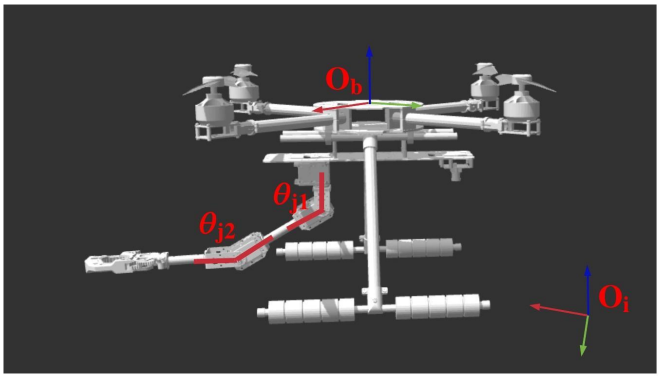


Figure 3.1: The geometric model of the UAM

3.2 Quadcopter Dynamic Model and Controller

We define state of the quadcopter as follows:

$$\boldsymbol{X_q} = [\eta, \omega, \xi, \nu]^T \tag{3.1}$$

where ω and ν are the angular and linear velocities in O_b, respectively. The body frame velocities are related to $\dot{\eta}$ and $\dot{\xi}$ as follows:

$$\omega = \begin{bmatrix} P \\ Q \\ R \end{bmatrix} = H\dot{\eta} \tag{3.2}$$

$$H = \begin{bmatrix} 1 & t(\theta)s(\phi) & t(\theta)c(\phi) \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi)/c(\theta) & c(\phi)/c(\theta) \end{bmatrix} \tag{3.3}$$

$$\nu = \begin{bmatrix} U \\ V \\ W \end{bmatrix} = R_i^b \dot{\xi} \tag{3.4}$$

$$R_i^b = \begin{bmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ c(\psi)s(\phi)s(\theta) - c(\phi)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\theta)s(\psi) & c(\theta)s(\phi) \\ s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) & c(\phi)s(\theta)s(\psi) - c(\psi)s(\phi) & c(\phi)c(\theta) \end{bmatrix} \tag{3.5}$$

where c, s and t are used for cos, sin and tan operators respectively, for the sake of readability.

Newton-Euler equations tells us the relation between the quadcopter state and the force and torque vectors applied to it at O_b, F_q and τ_q respectively, as follows:

$$\tau_q = J\dot{\omega} + (\omega \times J\omega) \tag{3.6}$$

$$F_q = m\dot{\nu} - g_b \tag{3.7}$$

where m is mass of the quadcopter, J is the inertial matrix of the quadcopter, and g_b is the gravity vector in O_b. If we solve the above equations for $\dot{\omega}$ and we get:

$$\dot{\omega} = J^{-1}(\omega \times J\omega - \tau_q) \tag{3.8}$$

$$\dot{\nu} = F_q/m - g_b \tag{3.9}$$

We also know how the propellers' rotational velocity relates to F_q and τ_q:

$$F_q = \begin{bmatrix} 0 \\ 0 \\ F_{q_z} \end{bmatrix}, \tau_q = \begin{bmatrix} \tau_{q_x} \\ \tau_{q_y} \\ \tau_{q_z} \end{bmatrix}, \Sigma = \begin{bmatrix} F_{q_z} \\ \tau_{q_x} \\ \tau_{q_x} \\ \tau_{q_x} \end{bmatrix} \tag{3.10}$$

$$C = \begin{bmatrix} c_t & c_t & c_t & c_t \\ 0 & dc_t & 0 & -dc_t \\ -dc_t & 0 & dc_t & 0 \\ c_q & -c_q & c_q & -c_q \end{bmatrix} U_q = \begin{bmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{bmatrix} \tag{3.11}$$

$$\Sigma = CU_q \tag{3.12}$$

where ϖ_i is the rotational speed of propeller i of the quadcopter, U_q is the vector of control inputs for the quadcopter, c_t and c_q are thrust and torque constants of the propellers, respectively, d is the distance between each propeller and the center of the quadcopter, and Σ is the vector of non-zero forces and torques applied to the quadcopter.

We used Symbolic Math Toolbox from MATLAB to solve the above equations in form of standard dynamic modeling:

$$\dot{X}_q = f_q(X_q, U_q) \tag{3.13}$$

The expanded version of this equations is available in appendix A.

3.2.1 Linear Model Predictive Controller (LMPC) for Attitude Control of the Quadcopter

As discussed in section 1.4.2 we chose MPC as the attitude controller for the UAM base. We define the attitude state of the quadcopter as follows:

$$X_a = \begin{bmatrix} \phi \\ \theta \\ \psi \\ W \end{bmatrix} \tag{3.14}$$

X_a includes four degrees of freedom of the quadcopter which can be controlled independently. By expanding the dynamic equations discussed in the previous section we see that calculation of $\dot{X}_a$ includes trigonometric functions of ϕ and θ while ψ and W appear in the linear part of the equations. Therefore, the dynamic model controlled by the attitude controller is nonlinear and the straightforward choice is to use a nonlinear model-predictive controller (NLMPC). However, NLMPC is computationally intensive and is not suitable for real-time control of a UAM. Our solution for this problem is to linearize the quadcopter around the hovering point and use a linear model predictive controller (LMPC) instead of NLMPC. Quadcopters generally go through small deviations from the hovering state (small ϕ and θ) so linearization at the hovering point gives a good approximation of their dynam-

ics. Note that this approximation breaks down for aggressive maneuvers that correspond to large changes in these parameters. In fact, we explicitly limit the value of these parameters via constraints in the LMPC formulation.

LMPC attempts to find the optimal control inputs, $\boldsymbol{U_q^*}$, within a receding horizon by minimizing a cost function that depends on the magnitude of control inputs, the rate of change of the control inputs, and error between the reference and current values of the system states formulated as follows:

$$\boldsymbol{U_q}^* = \arg\min_{U} \sum_{i=1}^{p} X_{q_i}^T Q X_{q_i} + \Delta U_{q_i}^T P \Delta U_{q_i} + U_{q_i}^T R U_{q_i} \tag{3.15}$$

where p is the prediction horizon, $\boldsymbol{U_q^*}$ is an array of control inputs with the length of p, ΔU is rate of change of the control inputs and Q, P and R are weight matrices. This optimization problem is subject to all the dynamic and kinematic constraints of the system. An iterative numerical optimization technique, based on Quadratic Programming, is used for minimizing the aforementioned cost function without violating the control and state constraints. Result of this solution is an optimal set of control inputs, $\boldsymbol{U_q^*}$. The first element of $\boldsymbol{U_q^*}$ is used as the control input of the system until a new $\boldsymbol{U_q^*}$ is available. We use Matlab MPC Toolbox [54] for the purpose of implementing this controller. Figure 3.2 depicts the attitude controller. In the following sections we will discuss how the attitude controller is used to follow the set-points given by the planner.

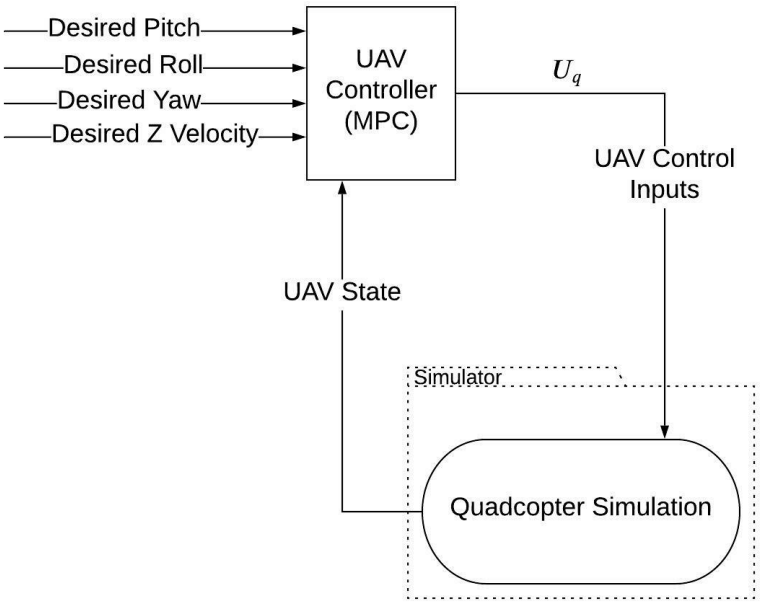


Figure 3.2: LMPC is used for attitude control of the quadcopter.

There are different parameters in LMPC which need to be tuned to achieve a good control performance. These are the main categories:

- Weights for cost of the control inputs and errors in following the reference values
- Control horizon
- Prediction horizon
- Control frequency

We used our simulation setup and ran extensive number of experiments to tune all of these parameters. Different values for these parameters changes the robustness and responsiveness of the controller. We want to have responsive controller that moves the state towards the set-points rapidly. At the same time, we don't want the controller to make the system unstable even if the given set points are oscillating. We gave different forms of the set points to the controller and picked the tuning that provided the fastest response without getting unstable in any of the cases.

3.3 Manipulator Dynamic Modeling and Controller

To achieve the dynamic model of the manipulator in the presence of of the quadcopter movements we used Recursive Newton Euler (RNE) equations. These equations are composed of forward propagation of the angular and linear velocities towards the end-effector and backward propagation of the forces and torques towards the base (quadcopter in our case). Please refer to [53] chapter 6 for details. We used the RNE equations considering a moving base to count for affects of the quadcopter movements, X_q and $\dot{X}_q$, on the dynamic of the manipulator. RNE backward propagation equations gives us two equations for the amount of torques exerted to the joints of the manipulator, τ_{j1} and τ_{j2}, in the following form:

$$\tau_{ji1} = f_{j1}(X_q, \dot{X}_q, \mathbf{\Theta}, \dot{\mathbf{\Theta}}) \tag{3.16}$$

$$\tau_{ji2} = f_{j2}(X_q, \dot{X}_q, \mathbf{\Theta}, \dot{\mathbf{\Theta}}) \tag{3.17}$$

$$\mathbf{\Theta} = \begin{bmatrix} \theta_{j1} \\ \theta_{j2} \end{bmatrix} \tag{3.18}$$

where θ_{j1} and θ_{j2} are the angles for the first and second joint of the manipulator, respectively.

We used Symbolic Math Toolbox from MATLAB to solve equations 3.16 and 3.17 for $\dot{X}_m$ and make it in the following form:

$$U_m = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) \tag{3.19}$$

$$U_m = \begin{bmatrix} \tau_{ji} \\ \tau_{j2} \end{bmatrix} \tag{3.20}$$

This model is used to develop a model-based nonlinear controller for the manipulator shown in figure 3.3

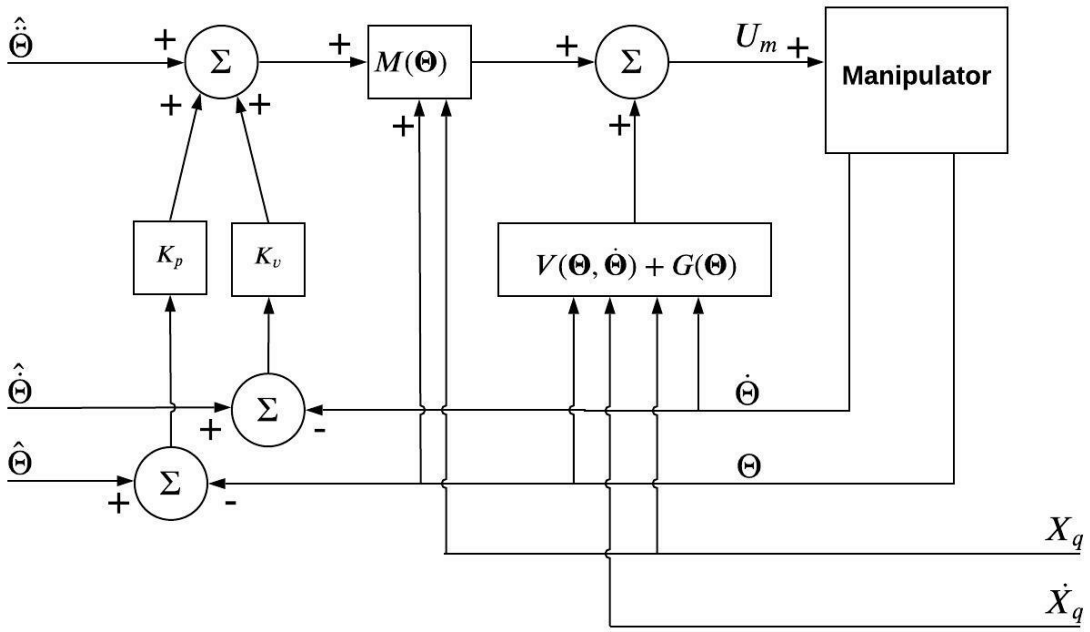


Figure 3.3: Model based controller for the manipulator.

3.4 Dynamic Effects of the Manipulator on the Quadcopter

The linear MPC design we proposed in section 3.2.1 is designed for the quadcopter without considering the manipulator. However, we know that manipulator movements affect the dynamics of the quadcopter. In the reminder of this section, we discuss how these effects are taken into account in order to control the quadcopter in presence of the manipulator.

We model the dynamic effects of the manipulator on the quadcopter using RNE equations as follows:

$$\tilde{\Sigma} = RNE(X_q, \dot{X}_q, \Theta, \dot{\Theta}) \tag{3.21}$$

where $\tilde{\mathbf{\Sigma}}$ is vector of torque and forces exerted to the quadcopter by the manipulator. This function takes states of both the quadcopter and the manipulator and returns $\tilde{\mathbf{\Sigma}}$ considering all the dynamic cross-effects.

We look at $\tilde{\Sigma}$ as a known disturbance exerted to the quadcopter controlled by LMPC. We propose a *cross-effects modeling* schema that keeps the quadcopter moving towards its set-point by adjusting the LMPC control policies, U_q, based on $\tilde{\Sigma}$. For this purpose, we first calculate $\mathbf{\Sigma}_{\boldsymbol{U_q}}$, the force vector generated by the LMPC control policy:

$$\mathbf{\Sigma}_{\boldsymbol{U_q}} = \boldsymbol{C U_q} \tag{3.22}$$

Then we add a force vector with the size of $\tilde{\Sigma}$ in the opposite direction to cancel out force vector generated by the manipulator:

$$\mathbf{\Sigma}_{\bar{\boldsymbol{U}}_{\boldsymbol{q}}} = \mathbf{\Sigma}_{\boldsymbol{U_q}} - \tilde{\mathbf{\Sigma}} \tag{3.23}$$

$\mathbf{\Sigma}_{\bar{\boldsymbol{U}}_{\boldsymbol{q}}}$ is the force vector we want to be generated by the adjust control policy. Therefore, we calculate adjusted control inputs, $\bar{\boldsymbol{U}}_{\boldsymbol{q}}$,as follows:

$$\bar{\boldsymbol{U}}_{\boldsymbol{q}} = \mathbf{\Sigma}_{\bar{\boldsymbol{U}}_{\boldsymbol{q}}} \boldsymbol{C}^{-1} \tag{3.24}$$

Figure 3.4 also depicts the equations for a better illustration of our proposed cross-effects modeling schema.

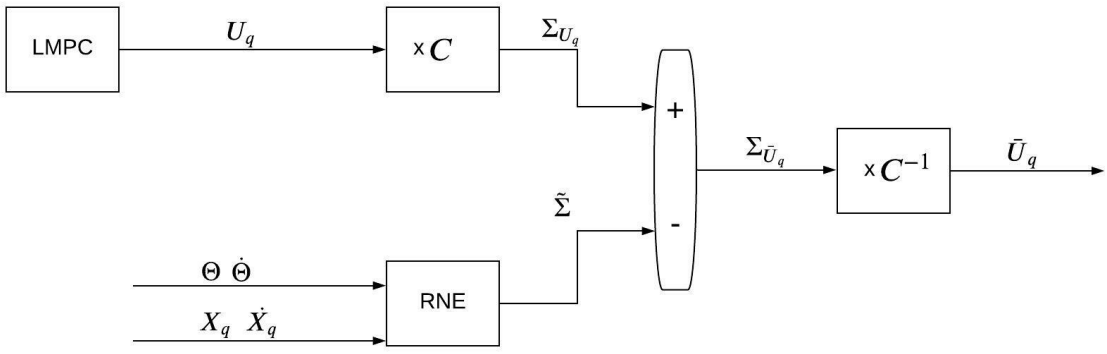


Figure 3.4: The proposed cross-effects modeling scheme.

3.5 Integrated Controller Schema

As mentioned above, we use separate dynamics of the quadcopter and the arm and control them with two different controllers, albeit considering all the cross-effects of the two subsystems. Figure 3.12 depicts the overall schema of our controller design which is used in both FAN and FBN phases to control the UAM. There are four major components in the proposed controller: position controller, attitude controller, arm controller and the cross-effects modeling module. *Position Controller* is a set of PIDs which generate attitude (roll and pitch) set-points based on the given X and Y set-points for the quadcopter. This step is needed as we are dealing with an underactuated system i.e. one can not control roll angle and Y position of the UAM independently. Although LS-RRT* plans in state-space and its solution includes velocity profiles and timing, we just enforce configuration of the robot (X, Y, Z, Yaw and arm's joint angles) and the controller follows these set-points in a closed-loop manner. As a result, timing of the trajectory in execution could be different from what is planned. *attitude Controller* is the LMPC discussed in section 3.2.1. *Arm Controller* is the arm controller we discussed earlier in this chapter. Finally, *cross-effects modeling* is the module that takes into account the effects of the arm motions on the UAV, discussed earlier in section 3.4.

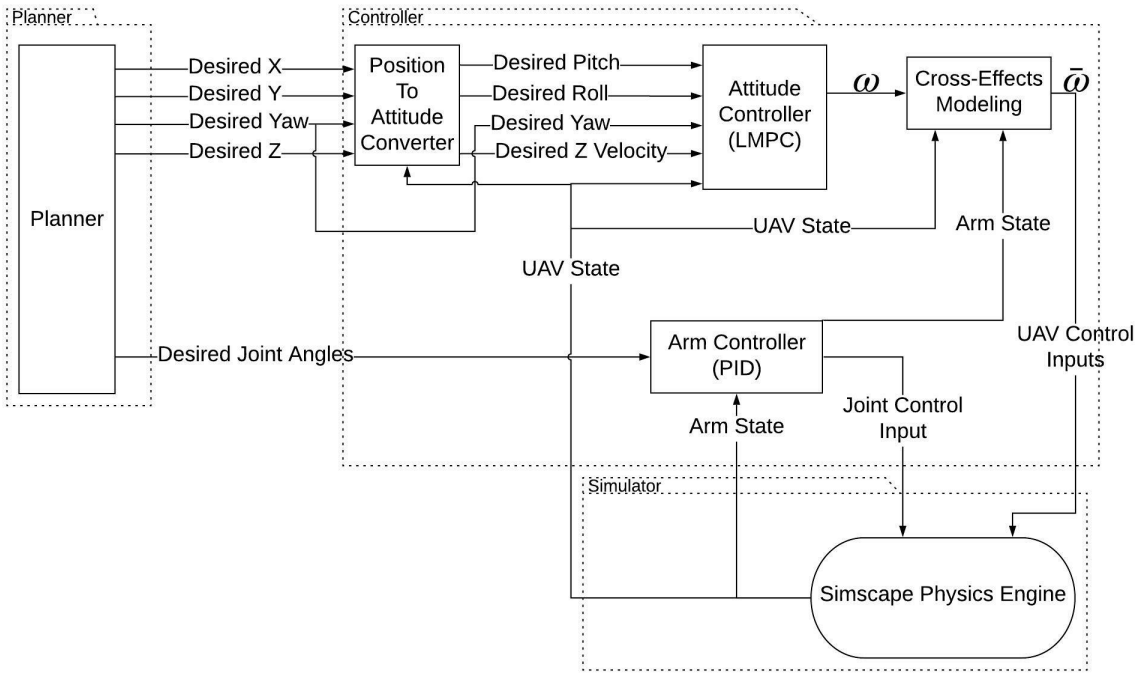


Figure 3.5: Partitioned controller schema.

3.6 Modeling Uncertainty

MPC is an optimal model based controller. This means that MPC is able to find the optimum control policy as long as the system model is accurate. Often the model of the system is not accurate; this error is called *modelling uncertainty*. Therefore, the main concern in designing a MPC controller is its robustness to modelling uncertainties. In robustness evaluation experiments we change the parameters of the quadcopter model used for simulation, namely the thrust and torque coefficients of the motors, to evaluate the robustness of the proposed controller to model uncertainties.
In the controller architecture we presented, we also used the model of the manipulator in cross-effects modeling scheme depicted in figure 3.12, in *cross-effects modeling* block. We add errors to the output of this block to observe performance of the controller in the case that estimating the manipulator force vector is not accurate. We will present the result of this evaluation at the end of this chapter in the results section.

3.7 Extension to Gain-Scheduled MPC

In this section we extend our proposed partitioned-yet-complete modeling and controller schema to a more complex UAM. Navig8 is a novel UAV system with pitch-hovering capability developed by 4FrontRbotics [1] depicted in Figure 3.6. Pitch-hovering is the capability of the UAV to stay steady while the pitch angle is not zero. This is not a possibility for the common UAVs such as quadcopters. Pitch-hovering feature of Navig8 makes its dynamic model more complex than a quadcopter. Out initial plan was to develop the full navigation pipeline for Navig8. Due to the Coronavirus situation complexities, however, further collaboration with 4FrontRbotics was not possible. As a result we changed the UAV to a quadcopter for rest of the work. In this chapter we present an optimal controller that enables the Navig8-manipulator system to follow a desired end-effector trajectory in the sagittal plane of Navig8, while the pitch angle varies with the Navig8 in a hovering position. Figure 3.7 depicts a schematic of such a task.

Navig8 has three rotors, two on the sides and a 3rd one with variable pitch propeller, on the tail. The side rotors are free to tilt around the lateral axis of Navig8. A key advantage of this three rotor design of the Navig8 is its pitch-hover capability. When Navig8 is equipped with a manipulator, as shown in figure 3.8, the reachability and maneuverability of the UAM is improved significantly by the pitch-hover feature of Navig8. It helps the UAM to perform a variety of missions (unachievable otherwise) such as landing on an inclined surface. However, this advantage comes at the expense of a more complex dynamic model, as explained in section 3.7.1.

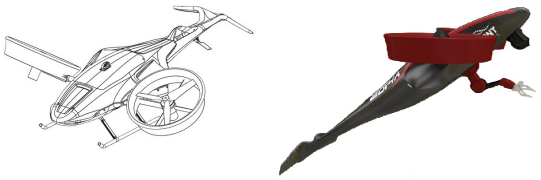


Figure 3.6: Navig8 is rotary wing UAV with pitch-hover capability produced by 4FrontRobotics [1].

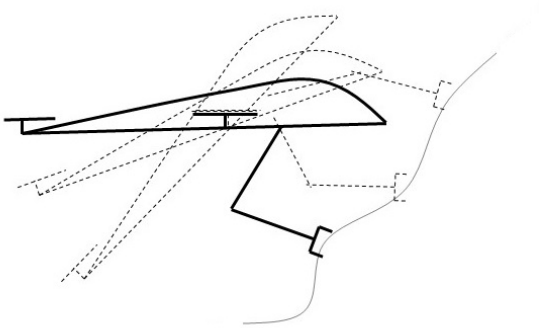


Figure 3.7: An example of end-effector trajectory following with pitch-hover.

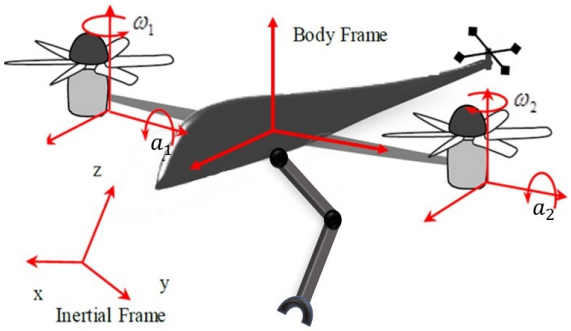


Figure 3.8: Navig8 equipped with a two link manipulator.

3.7.1 Modeling of Navig8

Figure 3.9 depicts the dimensions of Navig8 (in top view), needed for the dynamic modeling of Navig8. The body frame of reference, denoted by O_b, is located in the middle of the side rotors at the same height as the propellers. All computations for modelling and controlling the UAM are done in this frame. d_{tail} is the distance between the tail rotor and the origin along the X axis of O_b. d_{wing} is the distance between each of the side rotors and the origin O_b, along the Y axis.

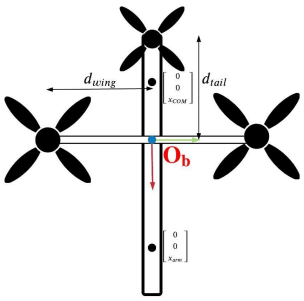


Figure 3.9: Navig8 from top view, showing kinematic parameters.

The standard Newton Euler equations of motion for Navig8 are as follows:

$$I_u \, \dot{\omega_u} = \bar{\tau}_n - \omega_u \times (I_u \, \omega_u) \tag{3.25}$$

$$m_u \, \dot{v_u} = \bar{F}_n \tag{3.26}$$

where I_u is the inertia matrix of Navig8 measured in O_b, m_u is the mass of Navig8 and ω_u and v_u are the angular and linear velocities of Navig8. One should note that in this thesis we use ω for the angular velocity vector of Navig8 and w (a scaler) for rotational speed of the rotors around the axis of rotation. $\bar{F}_n$ and $\bar{\tau}_n$ are the total force and torque exerted on Navig8. Although I_u changes with the rotation of the side rotors of Navig8, due to the small mass of the rotors and their slow rotation, we assume it is constant in this work. Our framework, though, is able to incorporate this variation, should it be needed, at the cost of getting a more complex dynamic model.

Figure 3.10 depicts all forces applied on Navig8. The total force and torque, $\bar{F}_n$ and $\bar{\tau}_n$, exerted on Navig8 consists of those generated by Navig8 rotors, F_n and τ_n, and the disturbances generated by the manipulator, $\tilde{F}_n$ and $\tilde{\tau}_n$:

$$\bar{F}_n = F_n + \tilde{F}_n \tag{3.27}$$

$$\bar{\tau}_n = \tau_n + \tilde{\tau}_n \tag{3.28}$$

F_n and τ_n are calculated by transferring the thrust and torque generated by each rotor to the body frame of reference O_b as follows:

$$F_x = -F_{r1} sin(a_1) - F_{r2} sin(a_2) + m \, g_x \tag{3.29}$$

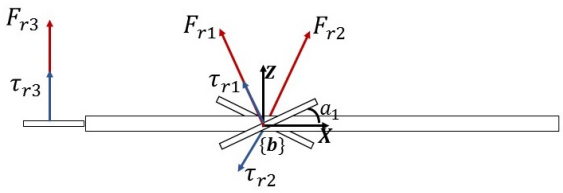


Figure 3.10: Fores and torques generated by the rotors, side view.

$$F_y = m\, g_y \tag{3.30}$$

$$F_z = F_{r3} + F_{r1}cos(a_1) + F_{r2}cos(a_2) + m\, g_z \tag{3.31}$$

$$\tau_x = d_{wing}F_{r2}cos(a_2) - d_{wing}F_{r1}cos(a_1) - \tau_{r1}sin(a_1) + \tau_{r2}sin(a_2) \tag{3.32}$$

$$\tau_y = d_{tail}F_{r3} - x_{com}m\, g_z \tag{3.33}$$

$$\tau_z = \tau_{r1}cos(a_1) - \tau_{r2}cos(a_2) + \tau_{r3} - d_{wing}F_{r1}sin(a_1) + d_{wing}F_{r2}sin(a_2) + x_{com}m\, g_y \tag{3.34}$$

where F_{ri} and τ_{ri} are the force and torque generated by the i^{th} rotor. $i = 1, 2$ refer to the right and left rotors, respectively, and $i = 3$ refers to the tail rotor. For the side rotors ($i = 1, 2$), the thrust and torque is calculated as follows:

$$F_i = c_{p_{side}} w_i^2 \tag{3.35}$$

$$\tau_i = c_{t_{side}} w_i^2 \tag{3.36}$$

where $c_{p_{side}}$ and $c_{t_{side}}$ are the thrust and torque constants of the side rotors and w_i is their rotational speed. The tail rotor is a variable pitch rotor with constant rotational speed. Dynamic equations of variable-pitch rotors is linearized and used to estimate the thrust and torque generated by the tail rotor as follows:

$$F_{t0} = c_{p_{tail}} w_{3_{hv}}^2 = x_{com} * m_u * g \tag{3.37}$$

$$\tau_{t0} = c_{t_{tail}} * w_{3_{hv}}^2 \tag{3.38}$$

$$F_{r3} = c_{p_{tail}} w_{3_{hv}}^2 \delta + F_{t0} \tag{3.39}$$

$$\tau_{r3} = c_{t_{tail}} * w_{3_{hv}}^2 \delta + \tau_{t0} \tag{3.40}$$

where $g = \begin{bmatrix} 0, & 0, & -9.8 \end{bmatrix}^T$, δ is the pitch angle of the tail rotor propeller, $w_{3_{hv}}$ is the constant rotational speed of the tail rotor which is selected such that Navig8 hovers with $\delta = 0$.

3.7.2 UAV Controller

As mentioned earlier, there are 5 actuators on Navig8, three rotors with propellers and two side tilt rotors. The tail rotor's rotational speed is constant but its generated thrust is adjusted by changing the propeller pitch angle. Therefore, we have five control inputs, $\begin{bmatrix} w_1 & w_2 & \delta & a_1 & a_2 \end{bmatrix}$, where w_1 and w_2 are the rotational speeds of the right and left side rotors, δ is the pitch angle of the tail rotor's propeller and tilt angles of the left and right side rotors. There are six degrees of freedom in Navig8, three for position of Navig8 and three for orientation. However, due to the under-actuated nature of the system, movements in y direction are not controlled directly. Instead we use roll rotations to control Navig8 along the y axis.

In contrast to quadcopters, Navig8 can have high pitch values even at hovering. For this reason, linearizing the dynamics at a single point is not accurate enough. To address this, we use LMPC [4] with implicit gain-scheduling to control Navig8. Navig8 dynamic model is linearized around a number of non-zero pitch angles for hovering. For real-time control, the region in which the Navig8 state resides is identified using a conventional estimation method at high frequency. The closest region is then looked up and predetermined linearized system parameters are used to calculate the control inputs via a standard MPC paradigm on the fly.

Our implementation of MPC is a velocity control. One could use MPC for position control, however, we empirically found that adding a structure behind MPC to convert the position references to velocity references improves convergence rate and solution quality of the controller. Figure 3.11 depicts the overall structure of the MPC controller and the position to velocity conversions. In this figure, superscript ˆon a variable denotes the desired value of the corresponding variable. Two PD controllers are used to convert the position and orientation errors to desired velocities for x, z, ϕ and ψ. MPC_{roll} is used to convert errors in y axis of Navig8 to desired roll rates. A common approach in the literature is to use a PID controller to generate desired roll rates. The PID output could be quite large in the case of a large error in y direction so a threshold is used after the PID controller to limit value of the desired roll rate. In the case of MPC, instead of thresholding the output, we set

explicit constraints for MPC optimization leading to a better performance. In experiments we found that using constrained MPC gives smoother optimal outputs.

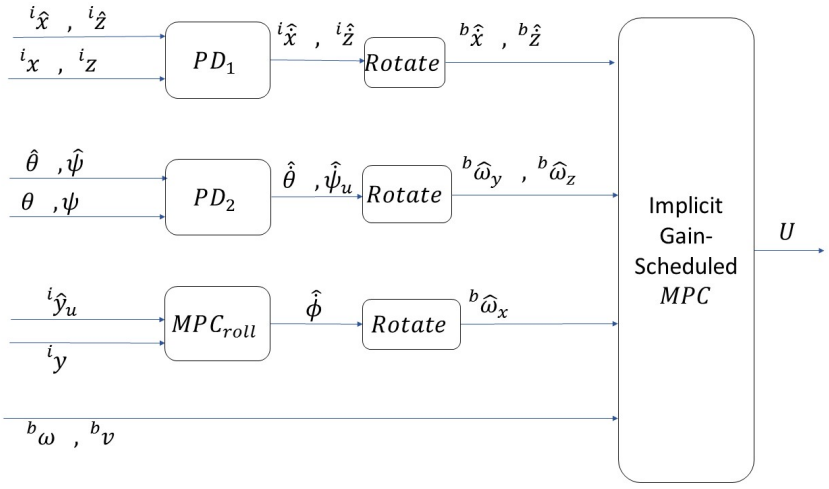


Figure 3.11: Model predictive controller design for UAV (Navig8).

As shown on the previous sections, the dynamic model of Navig8 is nonlinear. For such a nonlinear system, one option is to use nonlinear MPC as in [19]. The drawback is that it increases the computation time needed for MPC optimization in each control iteration and hence decreases the maximum frequency of the controller. In [19] the maximum achievable frequency for nonlinear MPC is $10Hz$ which does not work for real-time control of a UAV. In this work we want to use MPC as an *optimal* controller in *real-time* to optimize the control efforts of Navig8. Therefore, we use gain-scheduled linear MPC instead. As discussed the desired yaw and roll angles are zero but the pitch angle changes in the range of -50 to $+50$ degrees. Therefore, the key nonlinearity of our system is due to the pitch angle. We use implicit gain-scheduled MPC to handle this nonlinearity. The UAV model is linearized for different pitch angles with 10 degree resolution and during the flight the appropriate linearized model is used within MPC based on the current pitch angle.

3.7.3 Cartesian-space Arm Controller

Figure 3.12 is adapted from [53] and depicts the Cartesian space controller we use for controlling the two link arm. $\hat{x}_e$ and $\hat{z}_e$ are the desired position of the end-effector in the $\{X-Z\}$ plane of the inertial frame, τ_{ji} is the torque vector applied to the ith joint and θ_{ji} is the ith joint angle. We use $K_p = I_2 * 100$ and $K_v = 2\sqrt{K_p}$ for all experiments. In this scheme, the Cartesian space error is used directly to compute appropriate torques for the joints. In this controller the RNE equations are solved for the joint torques to write the arm dynamic model in this form:

$$\tau_e = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta) \tag{3.41}$$

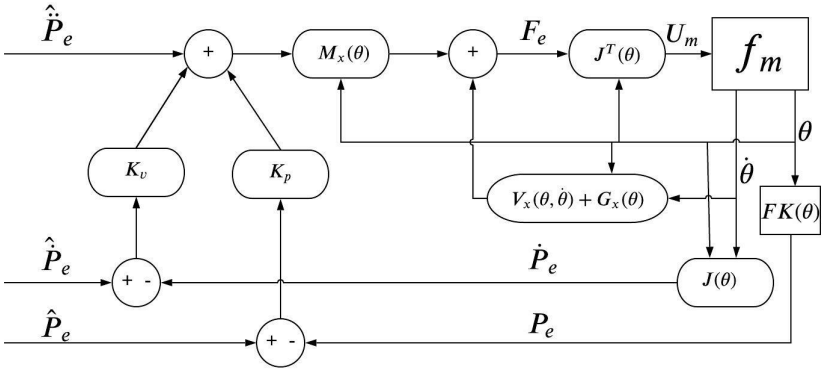


Figure 3.12: Cartesian space arm controller.

The arm dynamic model is also expressed in the Cartesian space as:

$$F_e = M_x(\theta)\ddot{\theta} + V_x(\theta, \dot{\theta}) + G_x(\theta) \tag{3.42}$$

where

$$\tau_e = J^T(\theta)F_e \tag{3.43}$$

, M is the inertia matrix of the manipulator, V is the vector of Coriolis and centrifugal terms and G is the vector of gravity terms. Please refer to [53] for more details on this controller.

3.8 Results

3.8.1 LMPC for Quadcopter based UAM

In this section we present the results of using the proposed decoupled control of the quadcopter-based UAM using LMPC. Specifically, we are interested to see if the proposed schema is capable of handling the dynamic affects of the manipulator movements on the base.

Figure 3.13 shows a graphical visualization of the proposed controller schema in following the planned trajectory for the UAV and the manipulator. The green and blue lines show the planned reference trajectories for the UAV and the end-effector, respectively. Note, our planner plans for the manipulator joints (configuration space) not for the end-effector (task-space). For visualization purpose in figure 3.13, the reference trajectory of the end-effector is calculated by forward-kinematic calculations. The purple and yellow curves show the actual traveled trajectory of the UAV and the end-effector, respectively. It can be seen that the UAM is following the set-points robustly.

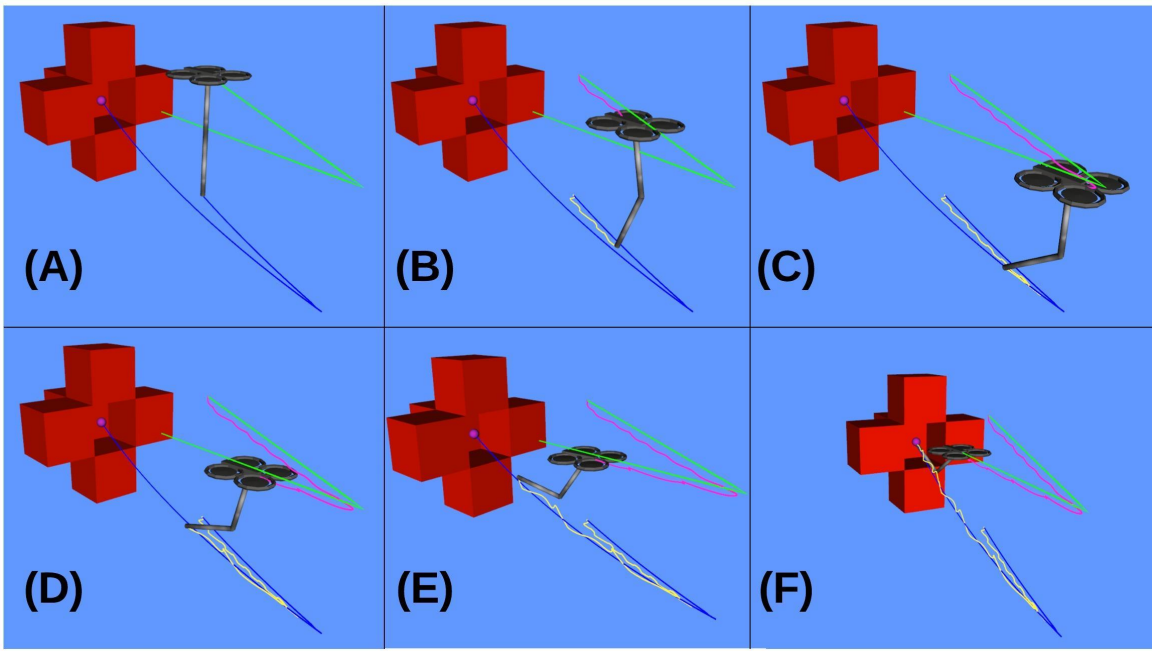


Figure 3.13: The reference and real trajectories of the UAV and the end-effector. Please see text for explanation.

Figures 3.14 and 3.15 show performance of the UAM and manipulator controllers, respectively. This is an example output of the extensive tests we performed. In these experiments, we use the planner for the FBN phase, Fast Marching Tree algorithm (FMT*) [35], to give set-points for all the configuration states of the UAM. As a result, both the UAM and the manipulator are moving simultaneously and the cross-dynamic effects are simulated by our Simscape model. It is observable in the results that the linear MPC controls the UAV through the set-point despite the fact that it has zero information about the manipulator in the dynamic model it uses. This shows that the cross effects, shown in figure 3.16, are handled well via our feed-forward scheme in our decoupled controller.

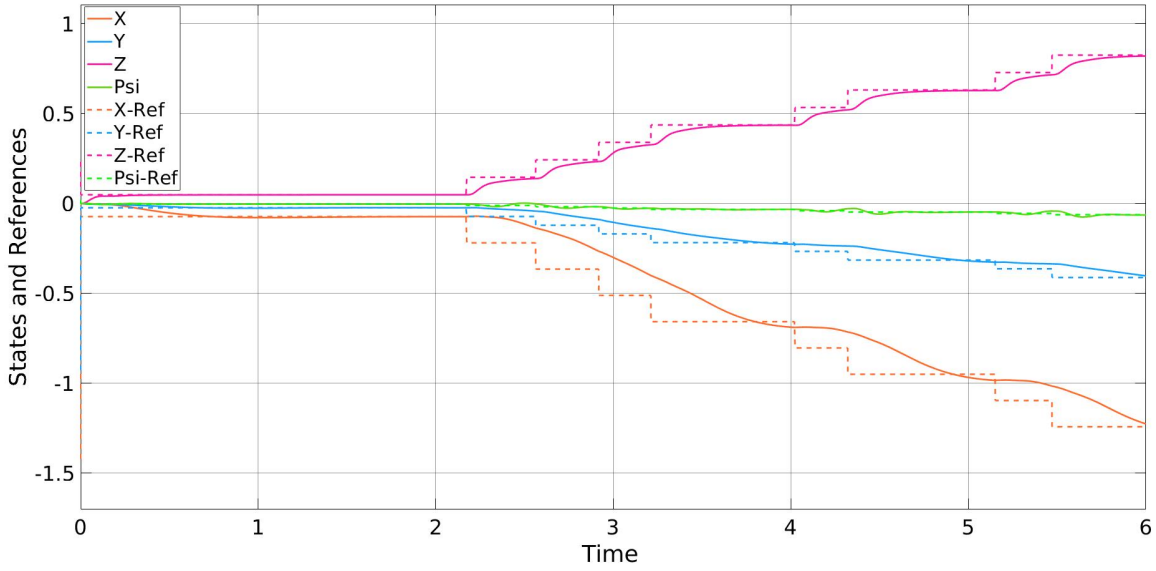


Figure 3.14: States of the UAV controlled by LMPC

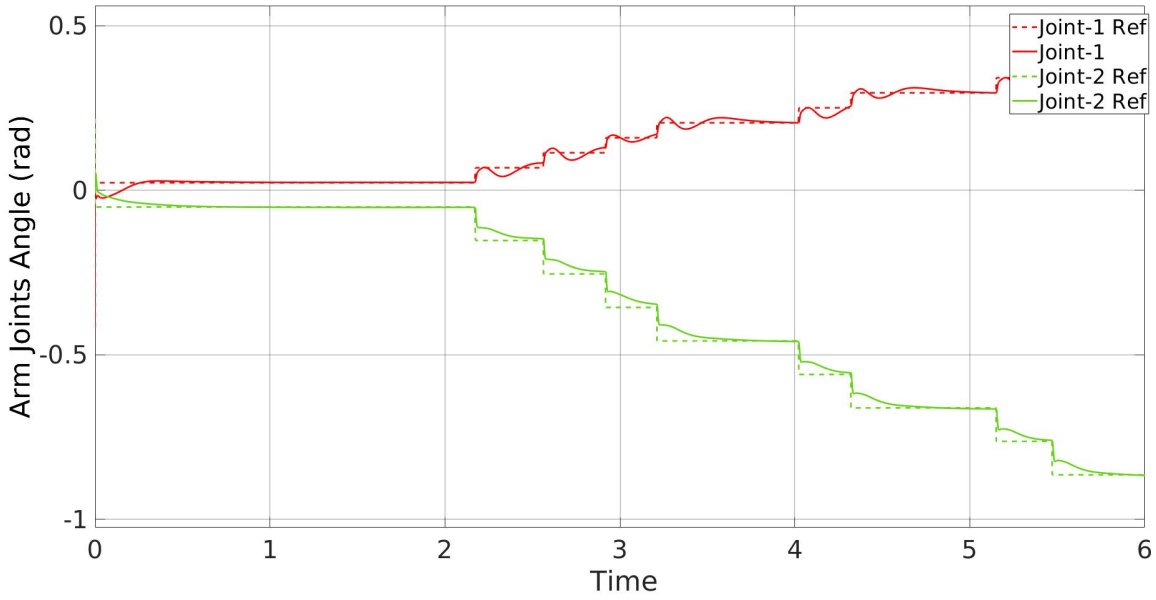


Figure 3.15: Arm joints angles controlled by the proposed arm controller

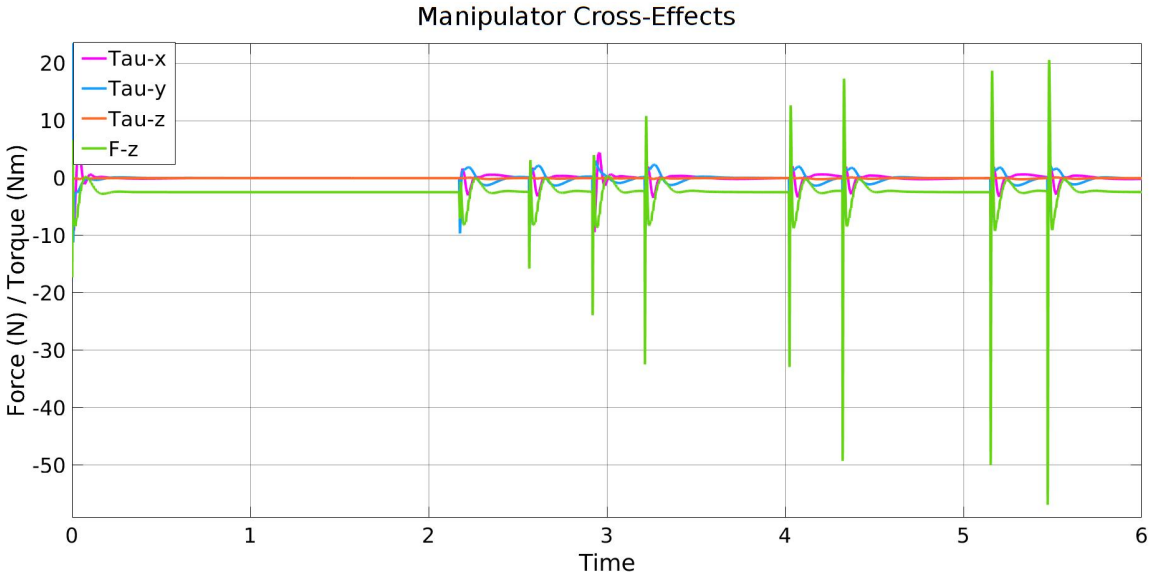


Figure 3.16: Cross-effects caused by movements of the manipulator

We also studied effectiveness of the proposed feed-forward "cross-effects modeling" in our partitioned controller extensively through simulation tests. The first group of tests were targeted to evaluate how the performance will be affected if we removed this block from the controller and provided the MPC policy directly to the UAV. Somewhat expected, our tests show that the UAM could be very unstable without considering the dynamic cross-effects depending on the magnitude of arm movements i.e. joints velocities and acceleration. For instance, in the case that the arm controller attempted to position the arm next to the object by reaching out to a grasping state, the UAV was not able to reach its own state-space set-points in absence of this block. A second group of tests were also carried out to evaluate tolerance of our schema to errors in the disturbance estimations - these errors can arise from modeling or sensing uncertainties. The schema was tolerant to errors of up to 8% in disturbance estimation, $\tilde{\mathbf{\Sigma}}$ in equation 3.4. Finally, to measure affect of noise in the acceleration measurements, we added simulated noise to the joints' acceleration measurements used in the RNE algorithm and found the system tolerant for up to 4% added noise. Sample results for these three categories of experiments are shown in figure 3.17

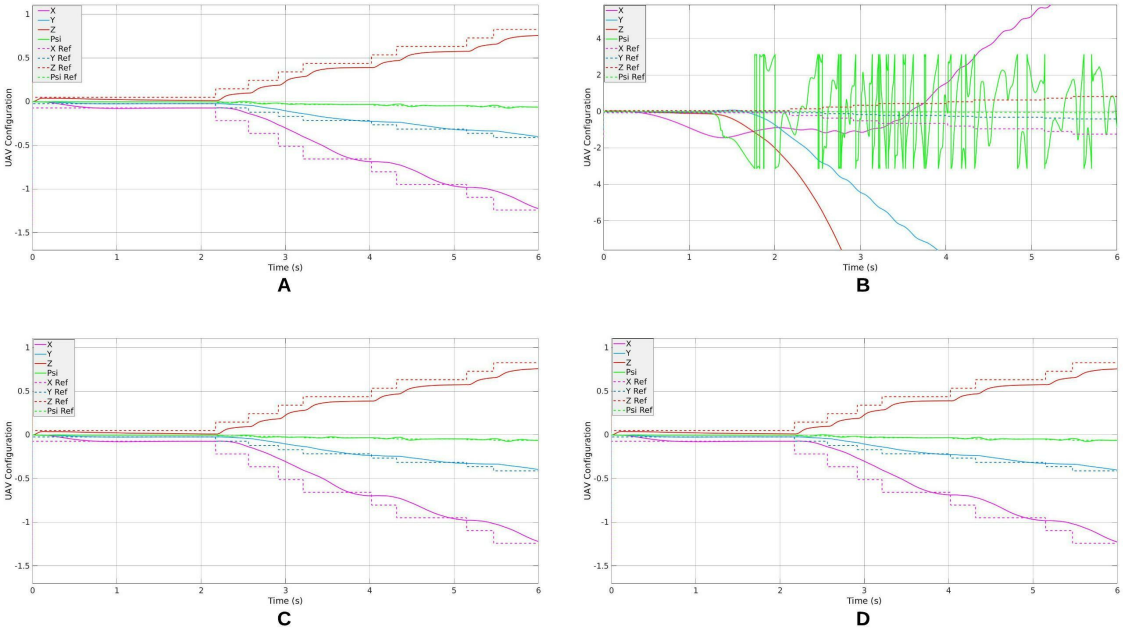


Figure 3.17: Controller performance (UAV state) in following the set-points in four cases A: with cross-effects modelling B: without cross-effects modelling. As one can see, system follows the set-point trajectory smoothly with cross-effects modelling (B) whereas without cross-effects modelling it is not stable (A). C: with 8% error in cross-effects calculation. The system is tolerant to up to 8% error. D: with 4% error in joints' acceleration measurements. The system is tolerant to up to 4% error.

$c_{p_{side}}$	$1.51e-5$	
$c_{t_{side}}$	$4.406e-7$	
$c_{p_{tail}}$	$8.51e-6$	
$c_{t_{tail}}$	$4.406e-8$	
$w_{3_{hv}}$	870.4848	rpm
m_{Navig8}	3.290	m
x_{com}	-0.2	m
I_{l1}	$\begin{bmatrix} 39401.08 & -69.07 & 624.65 \\ -69.07 & 1852809.26 & 12.72 \\ 624.65 & 12.72 & 1839381.82 \end{bmatrix} * 1e-9$	$kg.m^2$
I_{l2}	$\begin{bmatrix} 562200.59 & 162635.63 & -20948.98 \\ 162635.63 & 2906603.39 & 29321.76 \\ -20948.98 & 29321.76 & 2935055.02 \end{bmatrix} * 1e-9$	$kg.m^2$
m_{l1}	0.35	kg
m_{l2}	0.34	kg
COM_{l1}	$[0.259; 0; 0]$	m
COM_{l2}	$[0.154; 0; 0]$	m
I_{Navig8}	$[0.150300; 00.20780; 000.0702]$	$kg.m^2$
d_{tail}	1	m
d_{wing}	0.5	m

Table 3.1: Simulation parameters in metric units.

3.8.2 Gain-Scheduled MPC for Navig8 based UAM

In this section we present the result of using the proposed controller for Navig8 using gain-scheduled MPC. To evaluate the proposed controller, we run trajectory following simulations. In each simulation, a 2D Cartesian space trajectory in the $\{X-Z\}$ plane of the inertial frame is given to the system to be followed by the end-effector. This trajectory is reachable for the arm considering pitch angle of Navig8 (which changes during the simulation) and it avoids the singularities of the system. To demonstrate the capability of the proposed controller to follow a trajectory with varying pitch angle for Navig8, we change Navig8's pitch angle during the experiment from 0 to 50 degrees. We also ran experiments with the pitch angle going from 0 to -50 degrees and confirmed the controller works in this case as well. Table 3.8.2 states the parameters we used in the experiments.

Figure 3.18 depicts the real and desired trajectories of the end-effector and 3.19 depicts the error of trajectory following. It is observable that the trajectory is followed with a short delay. The maximum error in following the trajectory by the end-effector is 0.044 m. Attached to this thesis is a video of the end-effector trajectory following simulation. Refer to appendix B for more details about the video.

Figure 3.20 depicts the movement of Navig8 where it was supposed to stay at the origin. In our application, acceptable error for the UAV position is within 5cm. In simulations the maximum error is $\leqslant$ 17mm hence it satisfies the desired accuracy by a large margin.

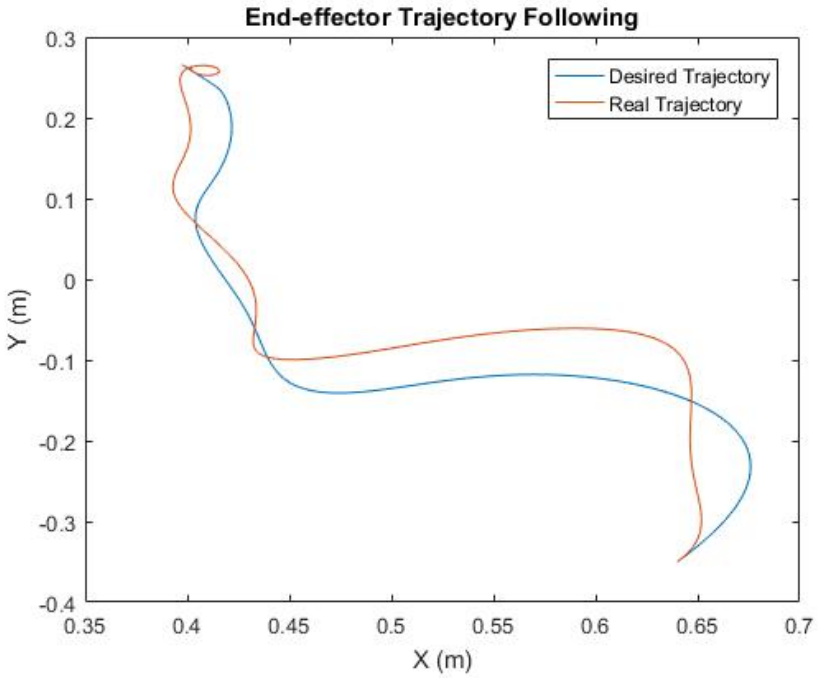


Figure 3.18: End-effector position; simulation result vs desired.

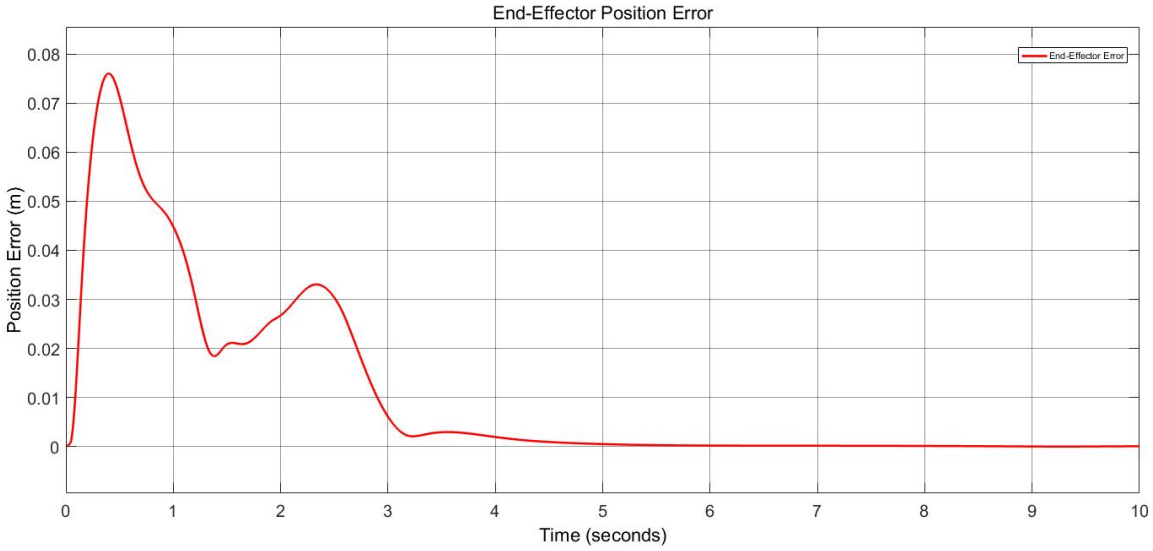


Figure 3.19: End-effector position error.

Simulations also show that in our approach, the optimal UAV controller (MPC) is able to run at $42Hz$ which is more than four times faster than [19]. We implemented the proposed controller using MATLAB and Simulnk version 2017 and run it on a machine with an "Intel Core i7 @ 3.40GHz" CPU. Our current code is somewhat inefficient in that it calls MATLAB functions within Simulink; getting rid of these inefficiencies would further increase

the controller rate. Of course, if implemented in C++ or Python, significantly increased rate could be obtained.

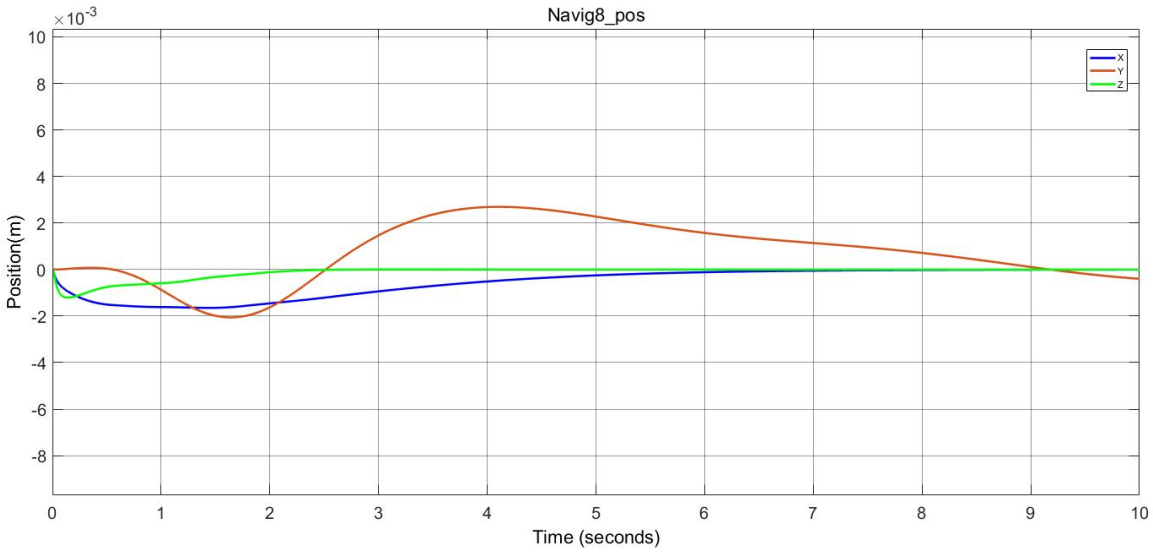


Figure 3.20: Error of Navig8 position controller.

Robustness Evaluation

As discussed in section 3.6, we also evaluated the robustness of our controller architecture to modelling uncertainties. We added modeling errors in two ways: i) we changed the thrust and torque coefficients of the UAV model used for simulation by up to 20%, and ii) we added errors of up to 30% to the manipulated disturbance estimated by RNE.
We then evaluate performance of the controller using two metrics, root mean square (RMS) and maximum of the end-effector position error. The results are tabulated in Table 3.2 and Figure 3.21 shows the end-effector trajectories for different parameter variations.

With up to 20% error in model parameters and up to 30% error in manipulator disturbance estimation, the maximum error in following the trajectory increases up to $0.0033m$ which is an increase of 4.34% compared to the ideal modelling scenario (experiment #1). This shows that the proposed controller scheme is robust to modelling uncertainties. One useful observation in these experiments is that overestimating the cross-effects is more problematic than underestimating them (compare experiments #2 and #3) while underestimating the UAV parameters (force and torque coefficients) is more problematic than overestimating them (compare experiments #5 and #6). Therefore, for better positional accuracy error in trajectory following, it is better to use lower bounds of the disturbance estimation and upper bounds of the force and torque parameters.

Table 3.2: Robustness evaluation

Experiment number	Modified UAV model parameters	Modified manipulator disturbance estimations	RMS	Maximum error
1	real parameters	RNE estimated values	0.0213m	0.076m
2	real parameters	RNE estimated values - 10%	0.0207m	0.0724m
3	real parameters	RNE estimated values + 10%	0.022m	0.0799m
4	real parameters	RNE estimated values - 20% error	0.0203m	0.069m
5	real parameters + 10%	RNE estimated values	0.0215m	0.0771m
6	real parameters - 10% error	RNE estimated values	0.0213m	0.0751m
7	real parameters ±10% (random)	RNE estimated values ±10% (random)	0.0211m	0.0748m
8	real parameters ±20% (random)	RNE estimated values ±30% (random)	0.0214m	0.076m

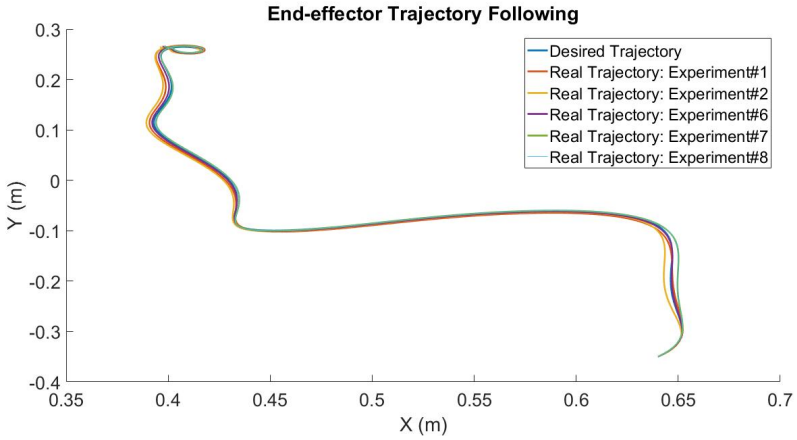


Figure 3.21: End-effector trajectory following results with modelling uncertainties.

Chapter 4

Integration And Evaluation

In this chapter we show how the proposed planning and control schemes are implemented and integrated into one system for object retrieval tasks along with an accurate simulation of the integrated system. Then we present the evaluation of the system in performing a set of realistic challenging object retrieval tasks.

4.1 SimScape Modeling

We used Simscape™ physic engine in Simulink® to simulate the dynamics of the UAM precisely. Our Simscape model, depicted in figure 4.1, is composed of the following parts: 1) quadcopter base 2) two arm links 3) two arm joints. Figure 4.2 visualizes the Simscape model. Table 4.1 include the parameters we used in the simulation.

In our implementation, the quadcopter has a mass of 1Kg and the distance between rotors to the center of the quadcopter is 22cm. Each link of the arm is 50cm and weighs 0.1 Kg.

Table 4.1: Parameters used in the SimScape model of the UAM

Parameter symbol	Description	Value
d	rotor to center distance	0.5 m
c_t	propeller thrust coefficient	1.4865e-07
c_q	propeller torque coefficient	2.9250e-09
m	quadcopter mass	1.0 Kg
J	quadcopter inertial matrix	$\begin{bmatrix} 0.0095 & 0 & 0 \\ 0 & 0.0095 & 0 \\ 0 & 0 & 0.0186 \end{bmatrix}$
m_{l1}	manipulator first link mass	0.1 Kg
m_{l2}	manipulator second link mass	0.1 Kg
l_{l1}	manipulator first link length	0.5 m
l_{l2}	manipulator second link length	0.5 m
J_{l1}	manipulator first link inertial matrix	$\begin{bmatrix} 3.9 & 0 & 0 \\ 0 & 185.2 & \\ 0 & 0 & 183.9 \end{bmatrix} 1e^{-5}$
J_{l2}	manipulator second link inertial matrix	$\begin{bmatrix} 56.2 & 0 & 0 \\ 0 & 290.6 & 0 \\ 0 & 0 & 293.5 \end{bmatrix} 1e^{-5}$

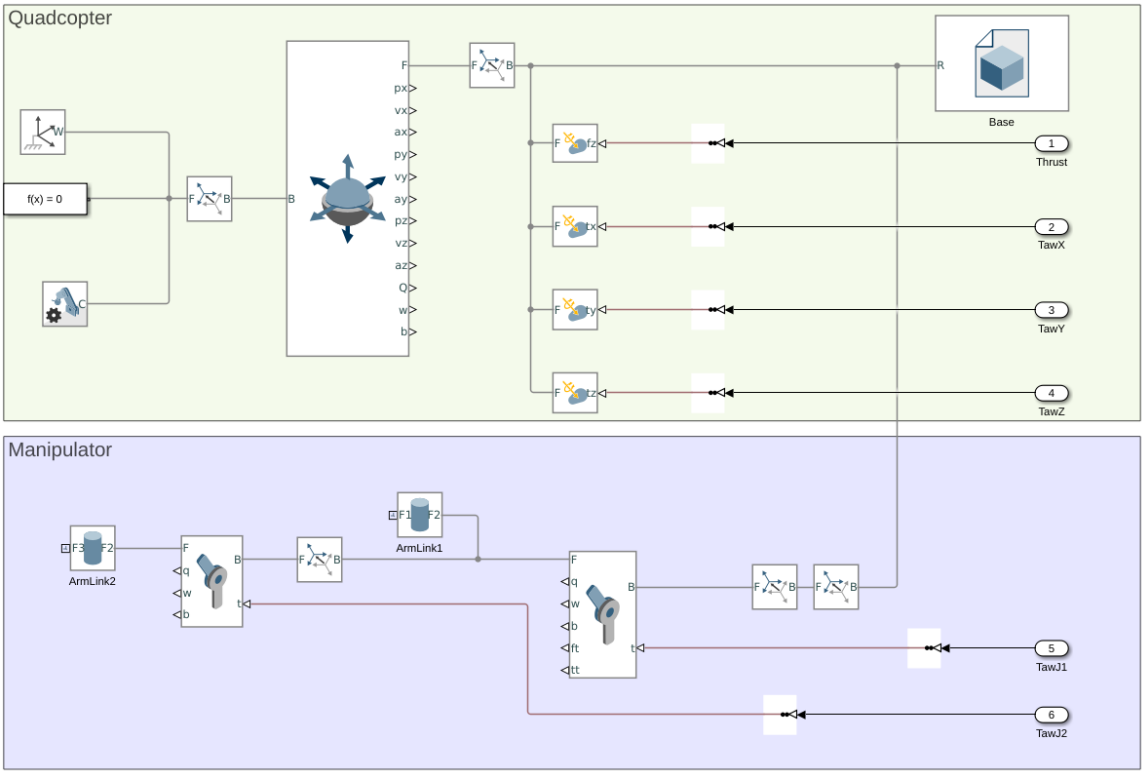


Figure 4.1: The high-level design of the Simscape simulation for the UAM.

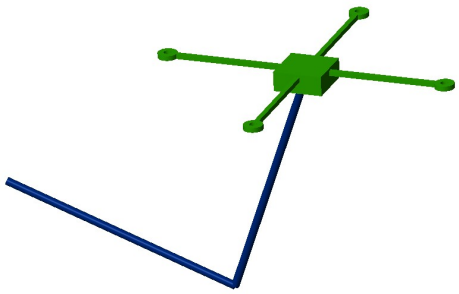


Figure 4.2: The geometric model of the UAM

4.2 MoveIt: Modeling UAM's Kinematics with a Serial Link Manipulator

MoveIt is a robust package used for motion planning of serial-link manipulators. We modeled simplified kinematics of the UAM in form of a serial-link arm so that we use MoveIt ROS for planning in the FBN stage. As discussed earlier, the UAM base does not move significantly during FBN and most of the actions are done by the manipulator. For this reason, we simplified the modeling by assuming zero pitch and roll angles. Using this assumption we are able to model kinematics of the UAM with three prismatic joints and three revolute joints as depicted in figure 4.3. We tried all the different planning algorithms provided by MoveIt and picked Fast Marching Tree algorithm (FMT*) [35]. FMT* is an optimal planner and gave us the highest success rate.

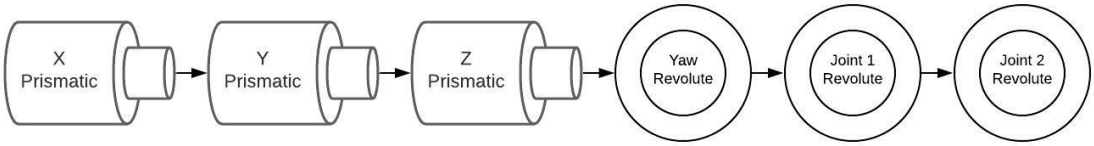


Figure 4.3: Kinematic modeling of the UAM in MoveIt

4.3 Modules Integration and Communication

In addition to MoveIt and SimScape modules discussed above, MATLAB was used to implement the proposed LS-RRT* planner and Simulink was used to implement the proposed partitioned controller. Also, we use Gazebo to create the obstacle world for experiments and a model for UAM is with the same kinematic specifications as in Simscape. All the components are connected to each other using ROS framework. Figure 4.4 and 4.5 depict the implementation design during FAN and FBN phases respectively. The main difference is that in FBN we don't use LS-RRT* and FMT* is used instead.

The initial state of the UAM, position of the object and the environment description is provided by the user. Gazebo simulation is used for visualization and is pure kinematic only.

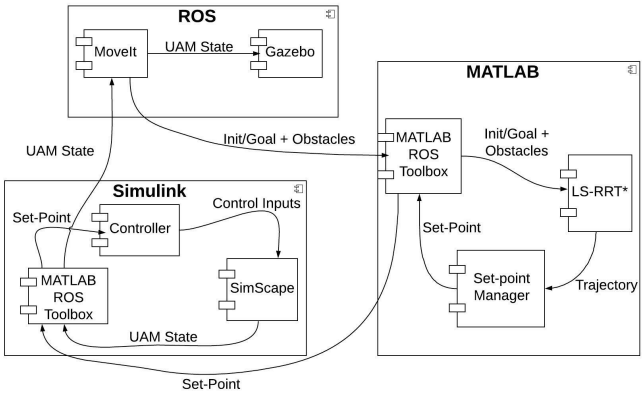


Figure 4.4: High-level system design in FAN phase

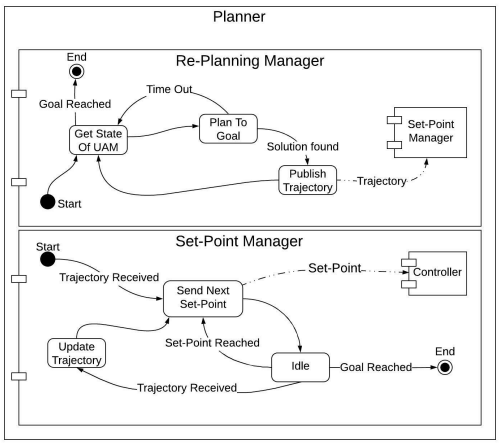


Figure 4.5: High-level system design in FBN phase

4.4 System Evaluation

To evaluate the proposed integrated navigation approach, we designed seven different worlds in Gazebo with different levels of complexity. Figures 4.6 - 4.12 show the worlds created in Gazebo. While, the planner assumes boundaries related to the floor and the ceiling, they are not shown in the Gazebo environment for sake of better graphical illustrations. In these scenarios, as shown in the figures, we designed the obstacles to represent different type of narrow passages such as windows, corridors and force the UAM to go through sharp maneuvers and also put the object in hard-to-reach places e.g. in a hole insert. Also, to

make the simulations realistic, we assume that sensors on the UAM can sense the obstacles only if they are within a 4m range from the UAM (shown by the green spheres in figures 4.13 and 4.14). Each of these scenarios were run several times with at least 5 different pairs of initial pose for the UAM and object positions. The reported numbers in this section are the average of all the runs for each scenario.

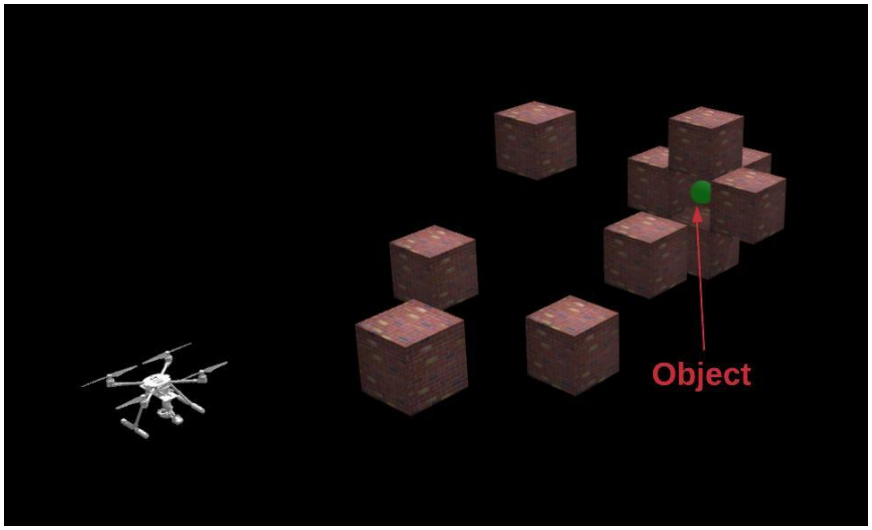


Figure 4.6: Simulation world for scenario 1.

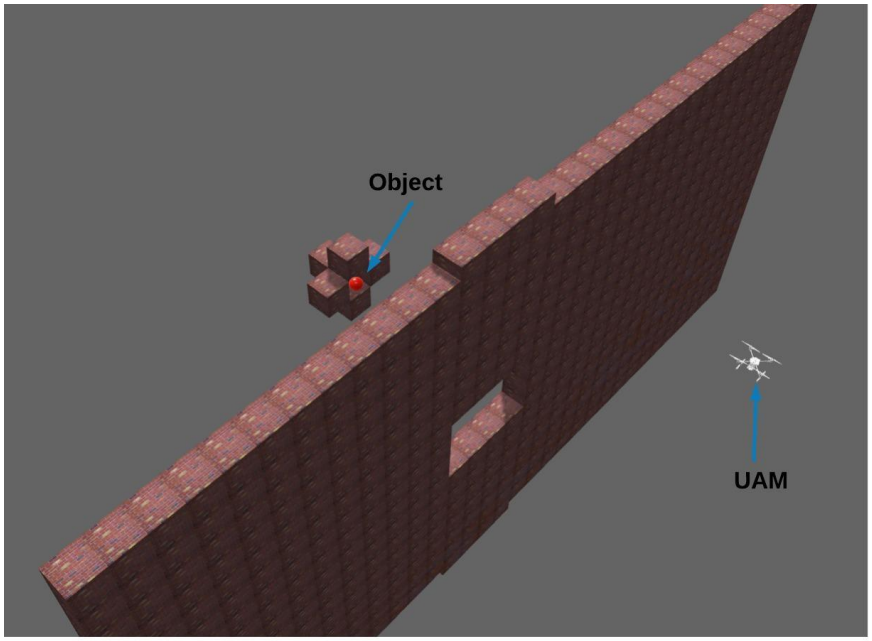


Figure 4.7: Simulation world for scenario 2.

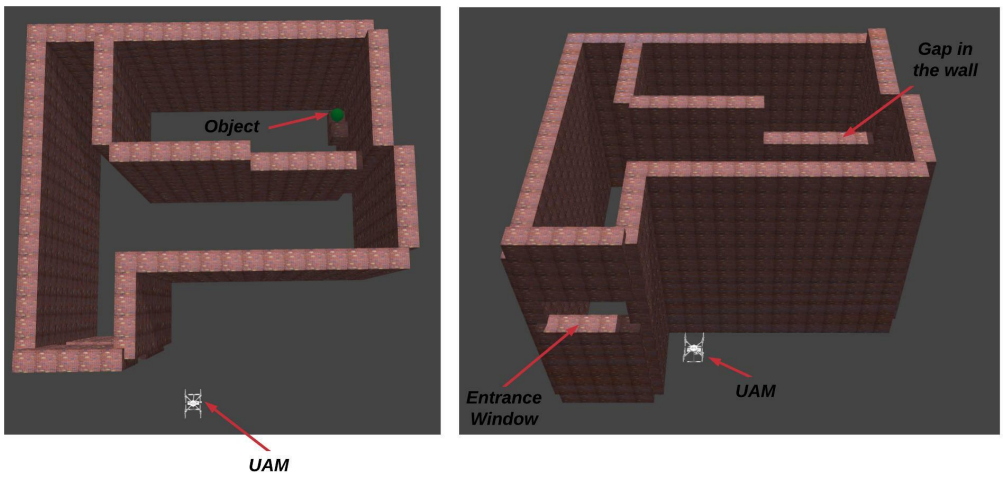


Figure 4.8: Simulation world for scenario 3.

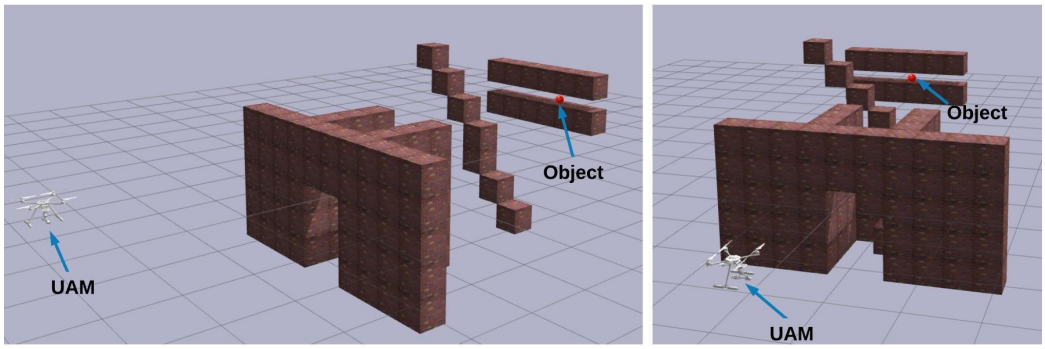


Figure 4.9: Simulation world for scenario 4.

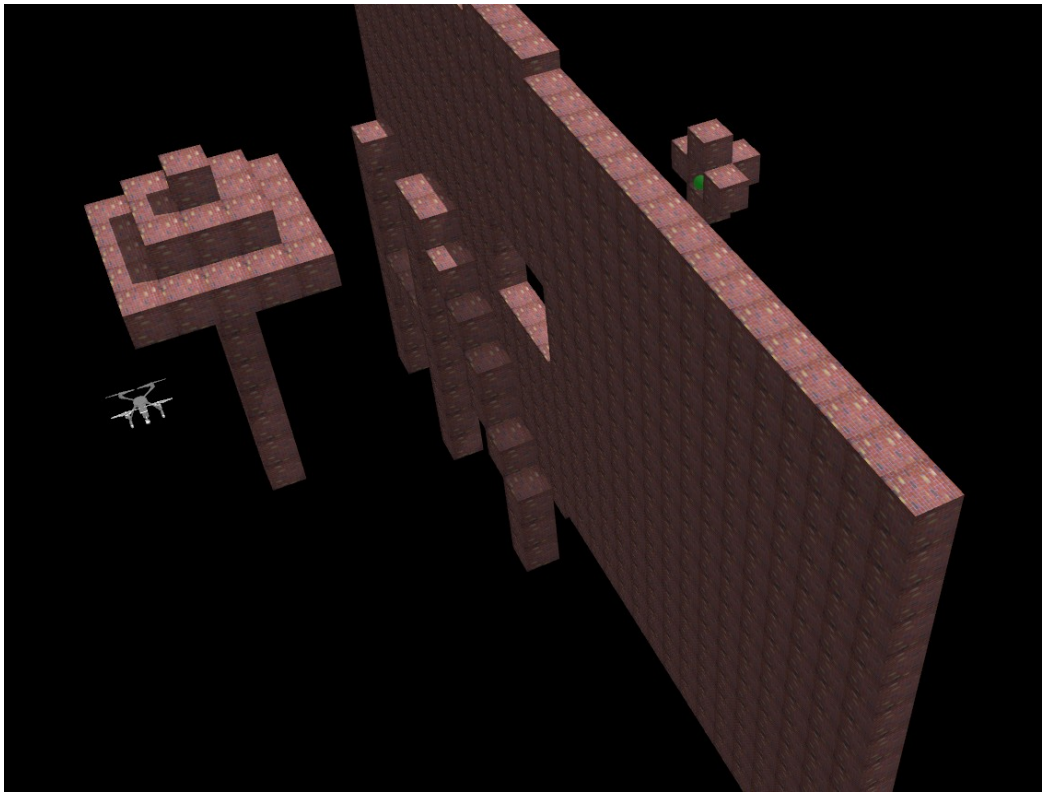


Figure 4.10: Simulation world for scenario 5.

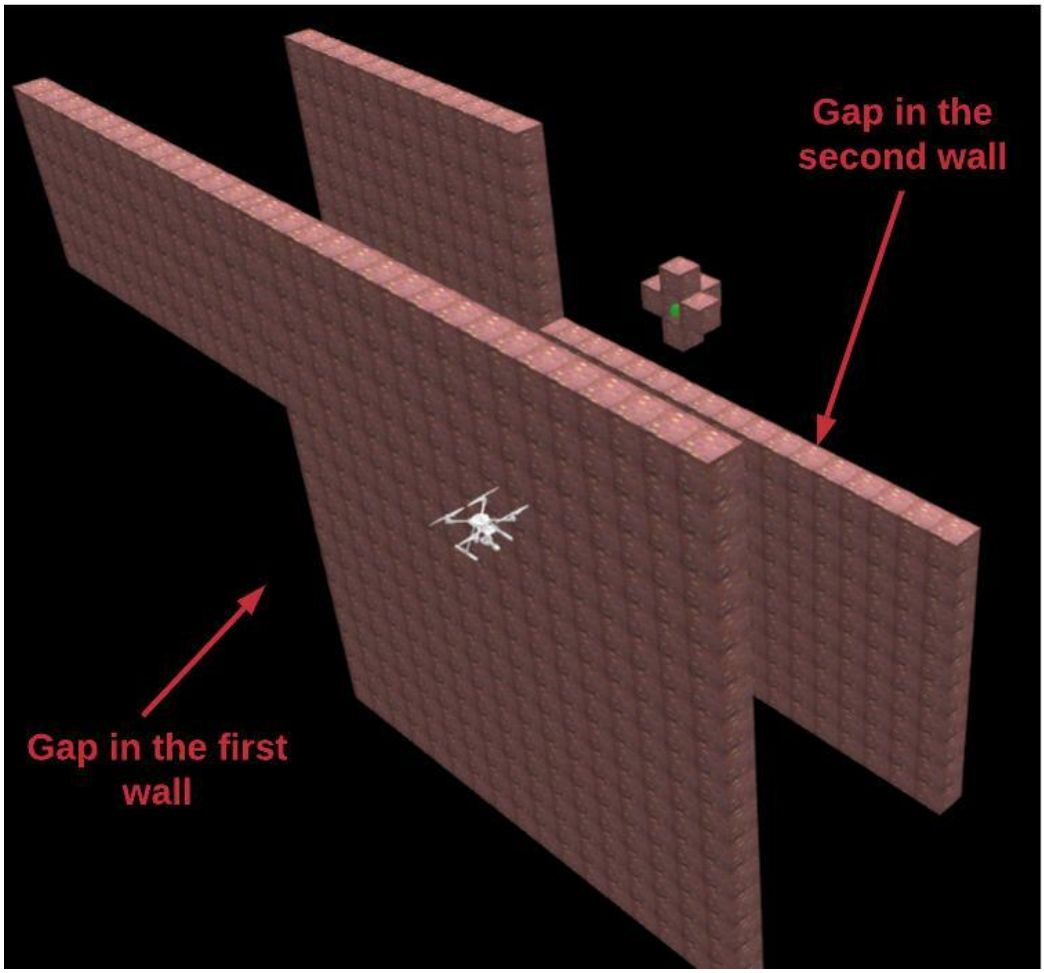


Figure 4.11: Simulation world for scenario 6. Two walls with opening on the opposite corners

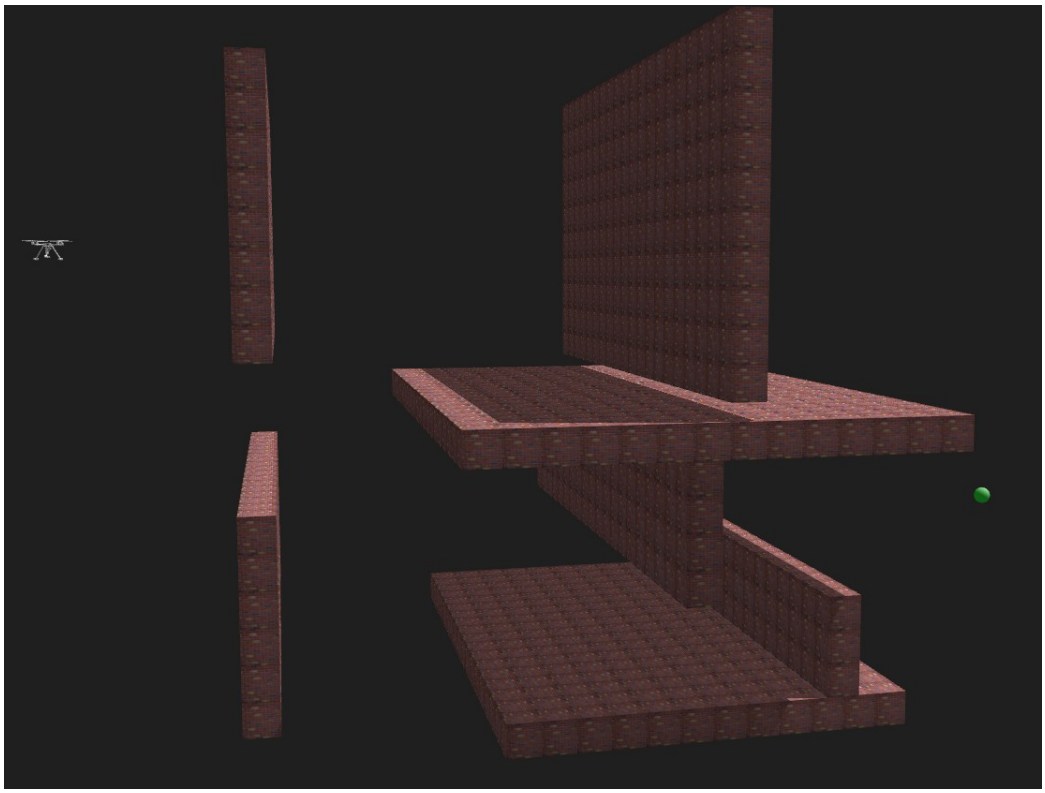


Figure 4.12: Simulation world for scenario 7.

We are interested in evaluating the integrated performance of the proposed controller and planner. We expect LS-RRT* to have fast run-time and to facilitate a high rate of re-planning during the FAN phase. With the replanning schema, we expect the UAM to smoothly react to new obstacles it detects. Furthermore, the controller performance is measured by how precisely it is able to follow the UAV set-points given by the planner during the FAN phase while keeping the manipulator folded, and follow the manipulator set-points to reach the object during the FBN phase while keeping the UAV at an in-position hovering state.

For the experiments reported in this chapter, we used a laptop with an Intel Core i7 9th Gen CPU, 16 GB of RAM, and a an NVIDIA GeForce GTX 1650 graphic card.

The planner performance is summarized in table 4.2. The second column of the table shows the time required (on average) for LS-RRT* to find the initial (non-optimal) path to the goal. In all of the scenarios, this number would be a fraction of a second. Having found the initial non-optimal path, LS-RRT* continues *rewiring* the search tree and improves the solution towards an optimal one. Here, as in all asymptotic-optimal anytime planners, there is a trade-off between optimality and the execution time. We empirically found that 15000 iterations will make a good balance between optimality and the run-time. The third column in the same table shows average run-time for one run of LS-RRT* with 15000 iterations.

Table 4.2: Replanning using LS-RRT*

Scenario	Average Number of Set-Points	First Solution Average Time (s)	Average Re-Planning time (s)	Space Volume (m^3)
1	13	0.21	3.49	135
2	21	0.15	3.79	315
3	23	0.13	4.73	300
4	24	0.24	4.52	300
5	31	0.22	4.15	270
6	26	0.18	4.64	280
7	42	0.31	4.85	350

This run-time ranged from 3.49s to 4.73s in our experiments. We believe that this can be further improved by migrating from an interpretation-type programming environment (as in MATLAB) to a compilation-based. According to the published studies [55], we can expect a speed-up of about 3 times under this shift in programming mode. For instance, we would expect a run-time of around 1.5s with a C++ implementation which means that a new plan will be available to execute only after a 1.5m motion in the UAM, given that it will be flying as fast as 1 m/s. In most of the applications, this will be quite reasonable. Also, it is worth noting that, in our experiments, a significant amount of the computation resources on the computer is used for running the simulation which is not needed in the case of real UAM deployment. We measured that on average a replaning run takes about 11% longer when the simulation is running.

Figures 4.13 and 4.14 depict different steps of the FAN phase for scenario 1 and 6, respectively. The green sphere around the UAM represents the simulated sensor range and field of view, i.e., how much information about the environment is available to the planner at each step. In figure 4.13 in step A, the walls in the building are not visible to the planner, hence the planned trajectory passes through them. Once the UAM is inside the building, in steps B and C, new sensor observations update the known map available to the planner and it can find a collision-free path through the available openings. The same change is observable in figure 4.14. In step A, the second wall is no fully sensible for the UAM so it plans through the wall. When more information is received, step B, a replanning iteration corrects the plan to avoid the obstacle.

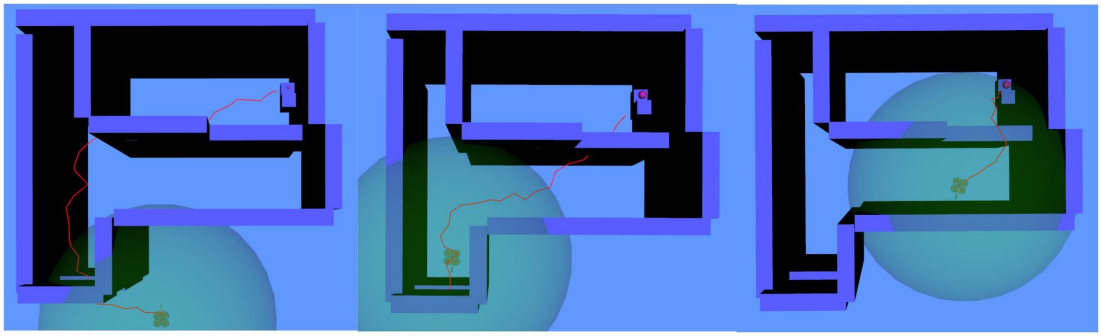


Figure 4.13: Three steps of FAN phase for scenario 3. The green sphere around the UAM shows limit of sensory information provided to the planner.

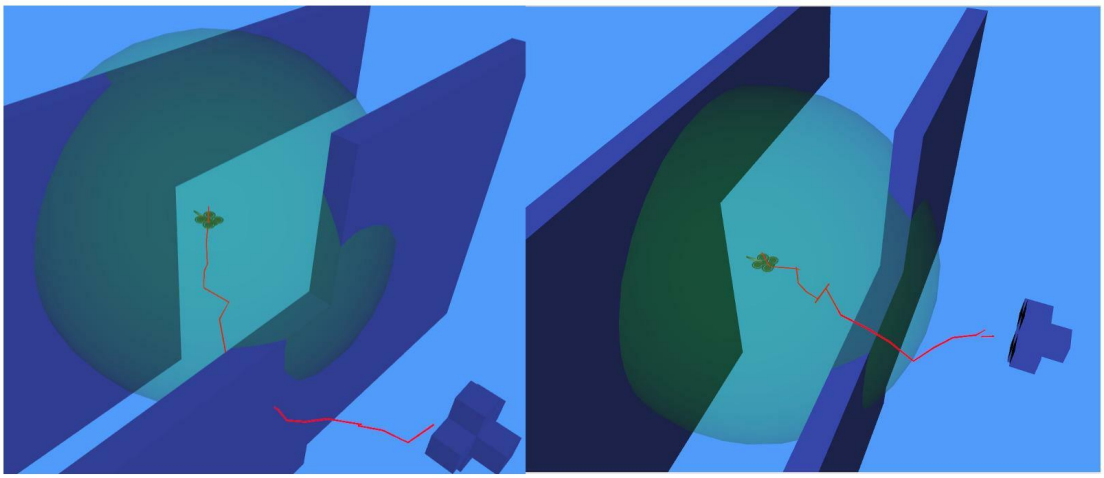


Figure 4.14: Two steps of FAN phase for scenario 6. The green sphere around the UAM shows limit of sensory information provided to the planner.

Figure 4.15 shows different steps of the UAM moving through scenario 2. The first three images are from the FAN phase and it is observable that the manipulator controller keeps the arm folded while the UAV controller (MPC) makes the UAM fly through the window following the set-points given by LS-RRT*. Steps 4 and 5 belong to FBN phase where the manipulator unfolds. In this phase the kinematic planner is giving set-points for the all DOFs of the whole-system, which are to be tracked by the partitioned controller up to the grasping point.

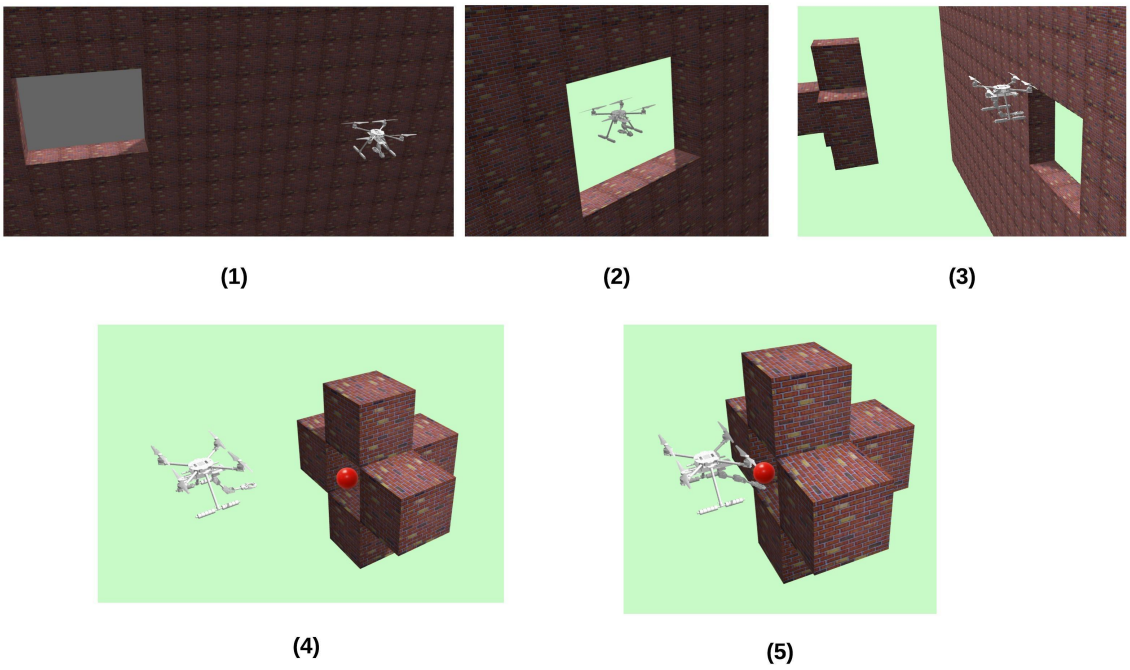
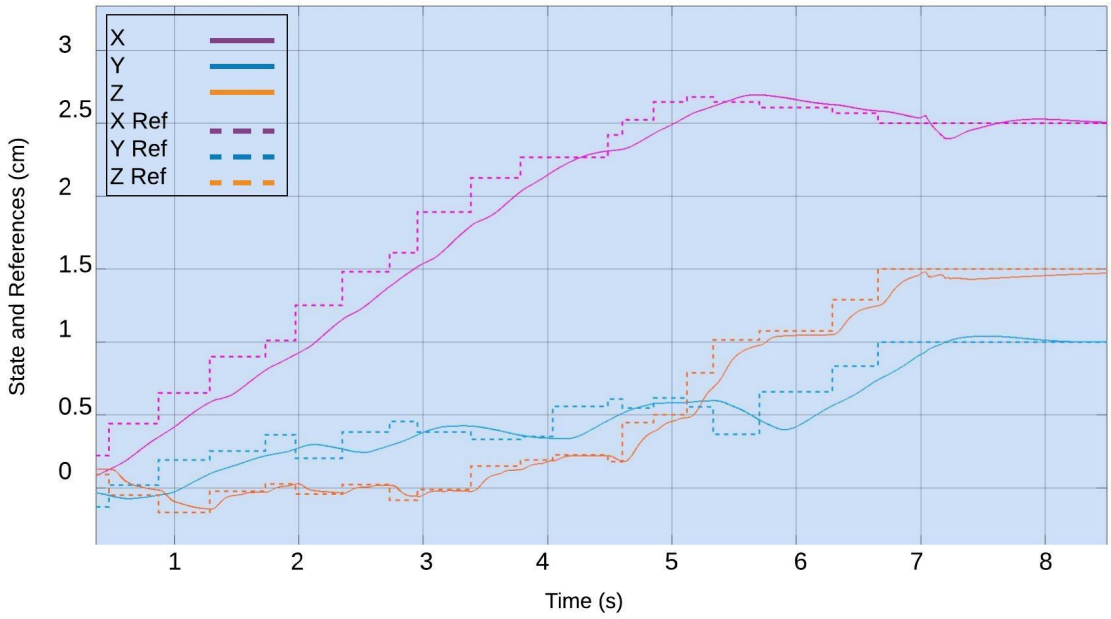
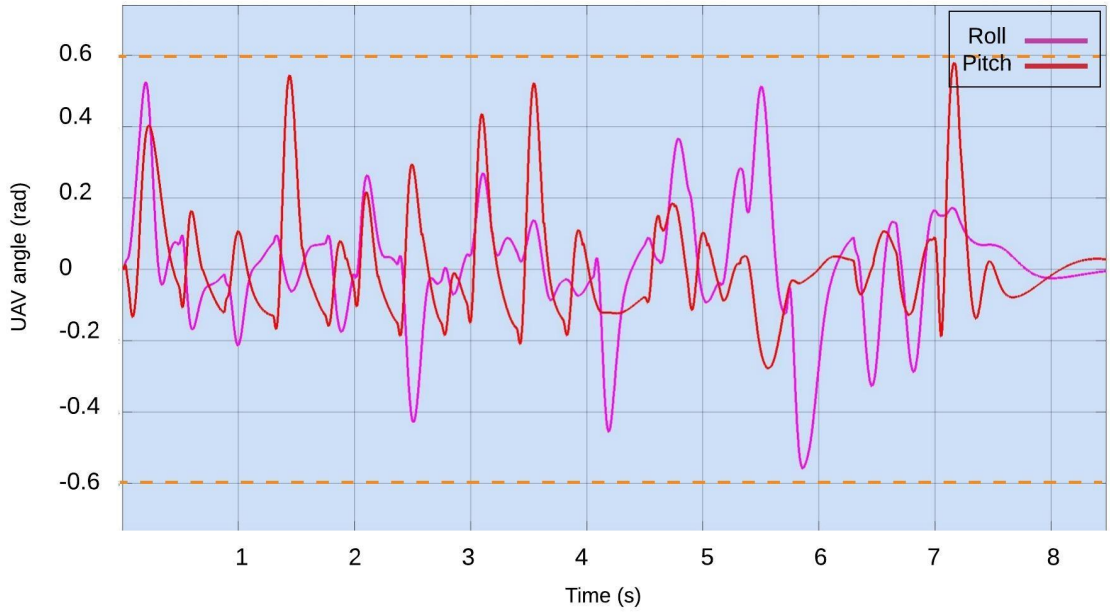


Figure 4.15: Screen shots of UAM trajectory in scenario 2.

Using the system mentioned above, the proposed controller ran at 385 Hz on average during our experiments. Figure 4.16 depicts the performance of the proposed controller in the FAN phase. In sub-figure (A on top, we see the transnational set-points i.e. X, Y and Z as dashed lines and the state of the UAM in solid lines. For the set-points in the FAN phase, we used a distance threshold of 0.1m for position tracking to determine if a set-point is reached, i.e., the next set-point is sent once the current state is within 0.1m of the current set-point. This was empirically chosen as a balance between accuracy and speed in following the set points To make sure that the UAM will not collide with obstacles, we add an inflation of 0.1m to the obstacles as a safety cushion. In sub-figure B, at the bottom, we see the time histories of the UAV pitch and roll angles. The orange dashed lines show the maximum limit of 0.6 radians that we set as a constraint for the roll and pitch angles in the quadcopter. Indeed, as can be seen in the figure, the controller is able to satisfy these constraints successfully.



A

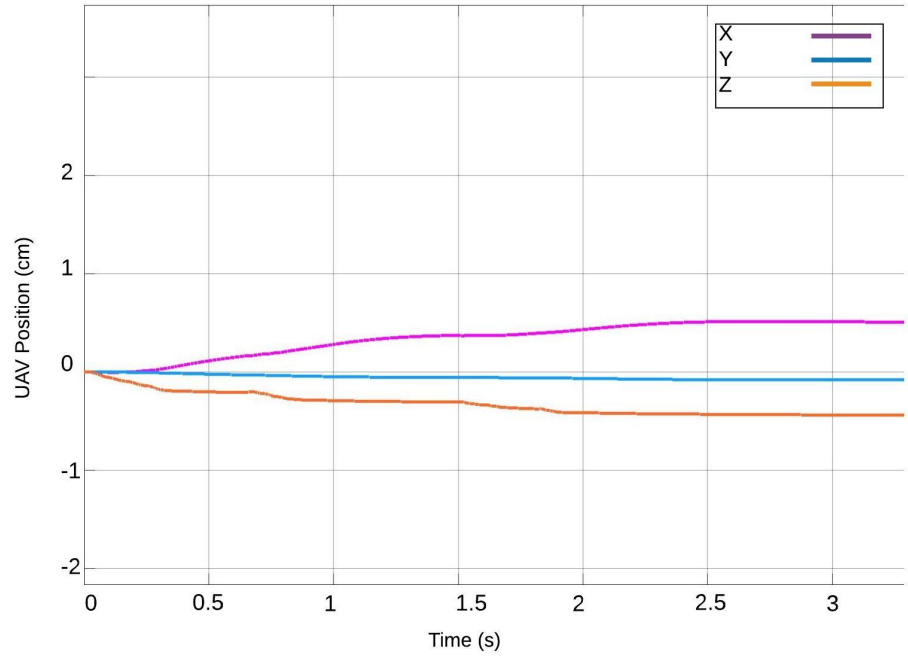


B

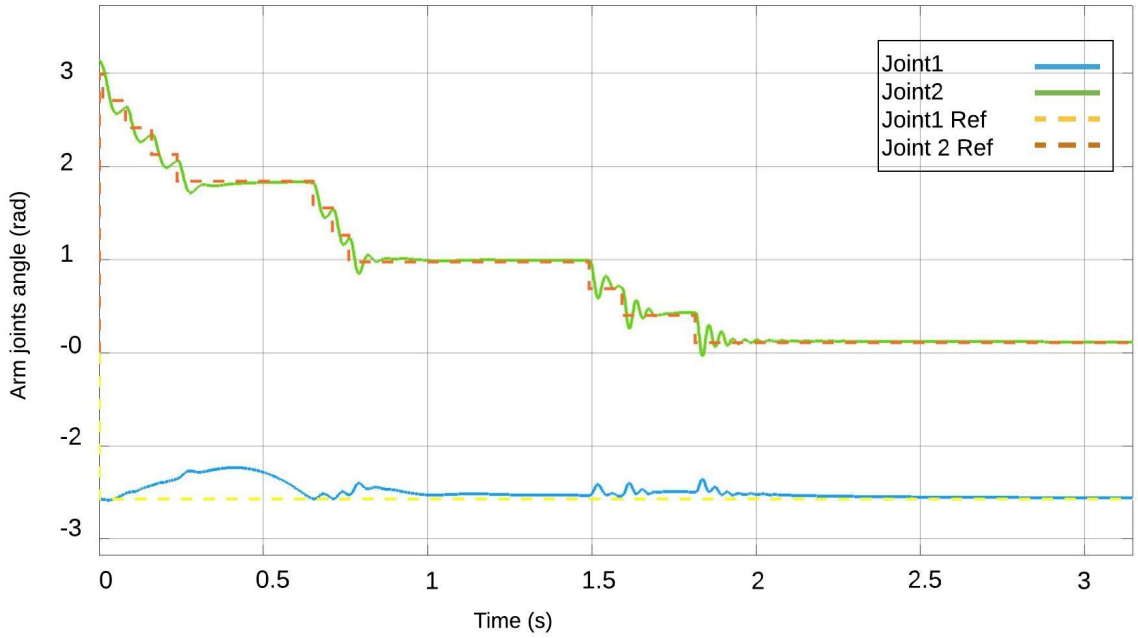
Figure 4.16: Controller performance (UAV state) in following the set-points during FAN phase in scenario 1

Figure 4.17 depicts trajectory of the UAM - UAV position (in sub-figure (a) on top) and arm joint angles (in sub-figure (b) at bottom) for the FBN phase (reference frame for position is set to point B i.e. initial position of FBN phase). One can see that the manipulator

is able to unfold and reach the object following the planned set-points while the quadcopter is moving minimally (less than 0.5m) during its hovering state, as desired. Indeed there is no significant fluctuation in the position of the quadcopter despite sudden changes in the configuration of the manipulator. This clearly demonstrates that our proposed method for estimating the cross-dynamic effects and modifying the control policies to compensate for them has been very effective and successful.



A



B

Figure 4.17: UAM state during FBN phase in scenario 1

Figure 4.18 depicts the minimum distance between the UAM and the obstacles during one of the runs in scenario 1. The distance to the obstacles gets smaller at the end where

the UAM approaches the object located in the hole pictured in Figure 4.15 (sub-figures 4 and 5) with a minimum distance of 2.7cm.

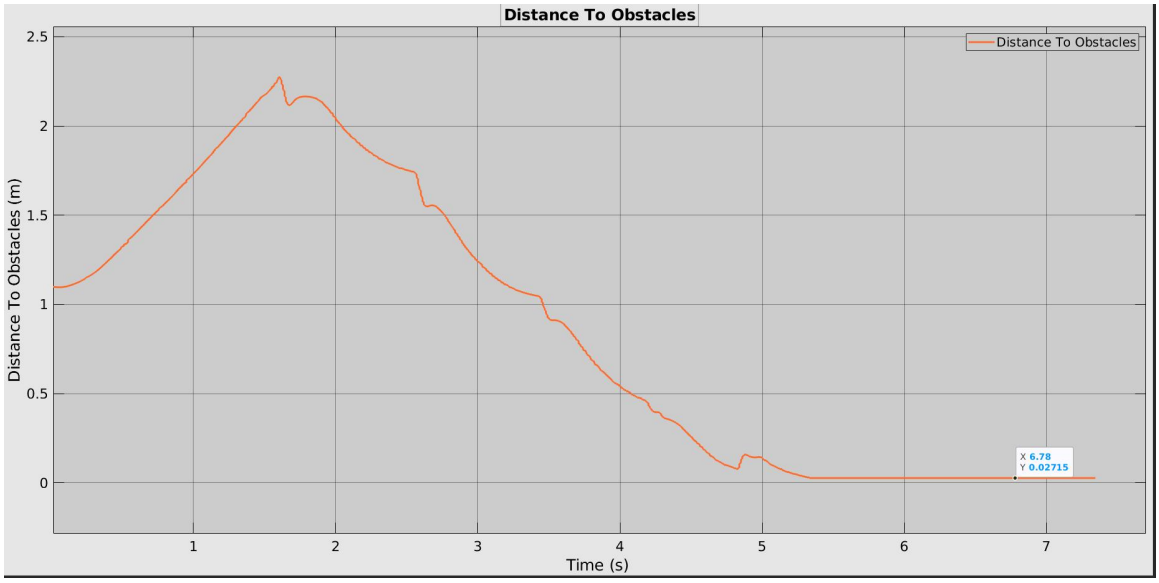


Figure 4.18: The minimum distance between the UAM and obstacles during scenario 1.

There is a video attached to this thesis that shows a sample run of the quadcopter-based UAM in scenario 3. Refer to appendix C for more details about the video.

Chapter 5

Conclusion

5.1 Contributions

We have proposed a novel integrated planning and control strategy for object retrieval with UAMs in unknown environments. The key core elements of the strategy are: i) a newly developed fast kinodynamic version of the RRT*, called Lazy-Steering-RRT*, by using Machine-Learning-based techniques to construct the edges in the search tree in a time-efficient way. This allows us to plan UAM's motion from its start to a pre-grasp state in near real-time and facilitates re-planning on-the-fly, as limited range sensors sense more of the environment. ii) an MPC/PID control approach is utilized for end-effector trajectory tracking based on a novel dynamic-effect partitioning approach between the arm and the base. The overall motion planning and control algorithms have been implemented in simulation on a quadcopter with a 2-link arm using Simscape Multibody from Simulink to get the most realistic results. Our results show that the approach is effective in successfully executing the object retrieval tasks in confined spaces.

There are several key advantages of our work over the existing ones. To the best of our knowledge, our integrated re-planning and control schema is the very first one proposed and implemented for a UAM. We avoid use of excessive dynamic simplification approaches such as assuming negligible dynamic cross effects between the UAV and the arm. Our partitioned approach is inherently modular in that it allows to substitute different arms and UAVs and does not require a specifically designed arm. Our planner incorporates the dynamic constraints of the system, and furthermore, does not assume that the controller will perfectly follow a planned path. Finally, we achieved a significant speed-up (two orders of magnitude) in planner run time by using ML in the kinodynamic planning as compared to other works.

5.2 Future Work

In near future, our next step is to adapt and implement the proposed planner and controller algorithms on a real system - a recently acquired hexacopter and a 4-DOF manipulator arm in our lab. This work will be complemented by grasp-planning which enables the real system to perform the object retrieval task. The grasp planner can replace the planner we used in the FBN phase and a dedicated controller should handle the gripper.

Another future work is to develop a criteria for changing the current plan. In this work, our main objective was to speed-up the replanning and we managed to achieve rapid replanning. Now the question is, do we want to change the plan every time a new one is available? Our experiments shows that switching between plans rapidly leads to undesirable behaviours such as jerks or back-and-forth. To mitigate this problem, a criteria should be designed to provide the system with a plan only if it is desirable.

5.3 Final Word

The role of robots in our life is getting more and more significant. In some industries robots are helping to overcome the worker shortage and in some other applications they help to eliminate human errors in sensitive tasks. Among different type of robots, UAMs are of a great importance as they enable us to reach places we can not without them. In this work we focused on improving practicality of UAMs. Using the proposed navigation pipeline, UAMs are able to operate without any a priori information about the environment. If manipulation is not needed, the UAM keeps the arm folded and navigates through the obstacles considering all the kinodynamic constraints of the robot; and when it is time to grab or drop an object, it is planned for the UAM base and arm to achieve the target position. We believe these are important steps towards having more practical UAMs that contributes to a wider range of applications.

Bibliography

[1] "4front robotics." `https://www.4frontrobotics.com/`. Accessed: 2017-08.

[2] F. Ruggiero, V. Lippiello, and A. Ollero, "Aerial manipulation: A literature review," in *IEEE Robotics and Automation Letters*, vol. 3, pp. 1957–1964, 2018.

[3] "Tevel aerobotics technologies ltd." https://www.tevel-tech.com/. Accessed: 2021-08.

[4] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: theory and practice—a survey," in *Automatica, 1989*, vol. 25, pp. 335–348.

[5] S. M. LaValle, J. J. Kuffner, B. Donald, *et al.*, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[6] M. Yavari, K. Gupta, and M. Mehrandezh, "Lazy steering RRT*: An optimal constrained kinodynamic neural network based planner with no in-exploration steering," in *IEEE International Conference on Advanced Robotics (ICAR)*, pp. 400–407, 2019.

[7] M. Yavari, K. Gupta, M. Mehrandezh, and A. Ramirez-Serrano, "Optimal real-time trajectory control of a pitch-hover UAV with a two link manipulator," in *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 930–938, 2018.

[8] "Simscape Multibody." https://www.mathworks.com/products/simscape-multibody.html. Accessed June 2021.

[9] "Robot Operating System." https://www.ros.org/. Accessed June 2021.

[10] "Gazebo." http://gazebosim.org/. Accessed June 2021.

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," in *The International Journal of Robotics Research*, pp. 846–894, 2011.

[12] D. J. Webb and J. v. d. Berg, "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints," in *Computing Research Repository (CoRR) in arXiv, 2012.*

[13] S. Filipov, I. D. Gospodinov, and I. Faragó, "Shooting-projection method for two-point boundary value problems," *Applied Mathematics Letters*, vol. 72, pp. 10 – 15, 2017.

[14] R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," in *AIAA Guidance, Navigation, and Control Conference*, p. 1374, 2016.

[15] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in RRT* for optimal kinodynamic planning," in *53rd IEEE Conference on Decision and Control*, pp. 3737–3744, 2014.

[16] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *IEEE International Conference on Robotics and Automation*, pp. 2537–2542, 2012.

[17] P. M. Bouffard and S. L. Waslander, "A hybrid randomized/nonlinear programming technique for small aerial vehicle trajectory planning in 3D," in *Planning, Perception and Navigation for Intelligent Vehicles (PPNIV), 2009.*

[18] J. GaryReid, D. E. Chaffint, and J. T. SilverthornJ, "Output predictive algorithmic control: Precision tracking with application to terrain following," *J. Guidance and Control*, vol. 4, no. 5, 1981.

[19] G. Garimella and M. Kobilarov, "Towards model-predictive control for aerial pick-and-place," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4692–4697, 2015.

[20] S. Kim, S. Choi, and H. J. Kim, "Aerial manipulation using a quadrotor with a two DOF robotic arm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4990–4995, 2013.

[21] R. Mebarki, V. Lippiello, and B. Siciliano, "Image-based control for dynamically cross-coupled aerial manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4827–4833, 2014.

[22] K. Kondak, K. Krieger, A. Albu-Schaeffer, M. Schwarzbach, M. Laiacker, I. Maza, A. Rodriguez-Castano, and A. Ollero, "Closed-loop behavior of an autonomous helicopter equipped with a robotic arm for aerial manipulation tasks," in *International Journal of Advanced Robotic Systems, 2013.*

[23] V. Lippiello and F. Ruggiero, "Exploiting redundancy in Cartesian impedance control of UAVs equipped with a robotic arm," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 3768–3773, 2012.

[24] V. Ghadiok, J. Goldin, and W. Ren, "Autonomous indoor aerial gripping using a quadrotor," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4645–4651, 2011.

[25] A. Jimenez-Cano, J. Martin, G. Heredia, A. Ollero, and R. Cano, "Control of an aerial robot with multi-link arm for assembly tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4916–4921, 2013.

[26] A. Khalifa, M. Fanni, and T. Namerikawa, "MPC and DOb-based robust optimal control of a new quadrotor manipulation system," in *European Control Conference (ECC)*, pp. 483–488, 2016.

[27] A. Jain and G. Rodriguez, "Diagonalized lagrangian robot dynamics," in *IEEE Transactions on Robotics and Automation*, pp. 571–584, 1995.

[28] G. Garofalo, F. Beck, and C. Ott, "Task-space tracking control for underactuated aerial manipulators," in *European Control Conference (ECC)*, pp. 628–634, 2018.

[29] D. Lunni, A. Santamaria-Navarro, R. Rossi, P. Rocco, L. Bascetta, and J. Andrade-Cetto, "Nonlinear model predictive control for aerial manipulation," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 87–93, 2017.

[30] D. D. Morrison, J. D. Riley, and J. F. Zancanaro, "Multiple shooting method for two-point boundary value problems," in *Communications of the ACM*, pp. 613–614, 1962.

[31] L. Palmieri and K. O. Arras, "Distance metric learning for RRT-based motion planning with constant-time inference," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 637–643, 2015.

[32] M. Bharatheesha, W. Caarls, W. J. Wolfslag, and M. Wisse, "Distance metric approximation for state-space RRTs using supervised learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 252–257, 2014.

[33] Y. Li, R. Cui, Z. Li, and D. Xu, "Neural network approximation based near-optimal motion planning with kinodynamic constraints using RRT," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 11, pp. 8718–8729, 2018.

[34] W. J. Wolfslag, M. Bharatheesha, T. M. Moerland, and M. Wisse, "RRT-CoLearn: towards kinodynamic planning without numerical trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1655–1662, 2018.

[35] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," in *The International Journal of Robotics Research*, pp. 883–921, 2015.

[36] H. Huang, A. V. Savkin, and W. Ni, "Online UAV trajectory planning for covert video surveillance of mobile targets," in *IEEE Transactions on Automation Science and Engineering*, pp. 4692–4697, 2021.

[37] G. P. Kontoudis and K. G. Vamvoudakis, "Kinodynamic motion planning with continuous-time Q-learning: An online, model-free, and safe navigation framework," in *IEEE Transactions on Neural Networks and Learning Systems*, pp. 3803–3817, 2019.

[38] J. Welde and V. Kumar, "Coordinate-free dynamics and differential flatness of a class of 6DOF aerial manipulators," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4307–4313, 2020.

[39] M. Tognon, E. Cataldi, H. A. T. Chavez, G. Antonelli, J. Cortés, and A. Franchi, "Control-aware motion planning for task-constrained aerial manipulation," in *IEEE Robotics and Automation Letters*, pp. 2478–2484, 2018.

[40] H. Seo, S. Kim, and H. J. Kim, "Locally optimal trajectory planning for aerial manipulation in constrained environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1719–1724, 2017.

[41] J. Thomas, J. Polin, K. Sreenath, and V. Kumar, "Avian-inspired grasping for quadrotor micro UAVs," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 55935, American Society of Mechanical Engineers, 2013.

[42] B. L'Espérance and K. Gupta, "Safety hierarchy for planning with time constraints in unknown dynamic environments," in *IEEE Transactions on Robotics*, pp. 1398–1411, 2014.

[43] S. Petti and T. Fraichard, "Partial motion planning framework for reactive planning within dynamic environments," in *Proc. of the IFAC/AAAI Int. Conf. on Informatics in Control, Automation and Robotics*, 2005.

[44] P. Ramon Soria, B. C. Arrue, and A. Ollero, "Detection, location and grasping objects using a stereo sensor on UAV in outdoor environments," in *Sensors*, 2017.

[45] M. Yavari, K. Gupta, and M. Mehrandezh, "Sampling based constrained motion planning for floating base manipulators using constraints driven alternative parameterization," in *International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 357–362, 2018.

[46] M. Yavari, K. Gupta, and M. Mehrandezh, "Interleaved predictive control and planning for an unmanned aerial manipulator with on-the-fly rapid replanning in unknown environments," in *IEEE Transactions on Automation Science and Engineering*, conditionally accepted 2021.

[47] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[48] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of computational and applied mathematics*, vol. 124, no. 1-2, pp. 281–302, 2000.

[49] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2951–2957, 2015.

[50] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, 2014.

[51] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[52] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3255–3260, 2006.

[53] J. J. Craig, *Introduction to robotics: mechanics and control*. Pearson Prentice Hall Upper Saddle River, 3 ed., 2005.

[54] "Optimization problem." https://www.mathworks.com/help/mpc/ug/optimization-problem.html. Accessed Dec 2021.

[55] S. Aruoba and J. Fernandez-Villaverde, "A comparison of programming languages in economics: An update," in *IEEE Transactions on Robotics and Automation*, 2018.

Appendix A

Expanded Equations Of Motion

The expanded version of equation 3.13 is as follows. We present each row of $\dot{X}_q$ in a separate equation to improve the readability:

$$\dot{\phi} = X_q[4] + X_q[6] * cos(X_q[1]) * tan(X_q[2]) + X_q[5] * sin(X_q[1]) * tan(X_q[2])$$

$$\dot{\theta} = X_q[5] * cos(X_q[1]) - X_q[6] * sin(X_q[1]) \tag{A.1}$$

$$\dot{\psi} = (X_q[6] * cos(X_q[1]))/cos(X_q[2]) + (X_q[5] * sin(X_q[1]))/cos(X_q[2])$$

$$\dot{P} = ((J[1,2]*J[3,3] - J[1,3]*J[3,2])*(c_t*d*U_q[1]^2 - c_t*d*U_q[3]^2 + X_q[4]*(J[1,1]*X_q[6]-$$
$$J[3,1]*X_q[4]) + X_q[5]*(J[1,2]*X_q[6] - J[3,2]*X_q[4]) + X_q[6]*(J[1,3]*X_q[6]-$$
$$J[3,3]*X_q[4])))/(J[1,1]*J[2,2]*J[3,3] - J[1,1]*J[2,3]*J[3,2] - J[1,2]*J[2,1]*J[3,3]+$$
$$[1,2]*J[2,3]*J[3,1] + J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1])+$$
$$((J[2,2]*J[3,3] - J[2,3]*J[3,2])*(c_t*d*U_q[2]^2 - c_t*d*U_q[4]^2 + X_q[4]*(J[2,1]*X_q[6]-$$
$$J[3,1]*X_q[5]) + X_q[5]*(J[2,2]*X_q[6] - J[3,2]*X_q[5]) + X_q[6]*(J[2,3]*X_q[6]-$$
$$J[3,3]*X_q[5])))/(J[1,1]*J[2,2]*J[3,3] - J[1,1]*J[2,3]*J[3,2]-$$
$$J[1,2]*J[2,1]*J[3,3] + J[1,2]*J[2,3]*J[3,1] + J[1,3]*J[2,1]*J[3,2]-$$
$$J[1,3]*J[2,2]*J[3,1]) + ((J[1,2]*J[2,3] - J[1,3]*J[2,2])*(-c_q*U_q[1]^2 + c_q*U_q[2]^2-$$
$$c_q*U_q[3]^2 + c_q*U_q[4]^2 + X_q[4]*(J[1,1]*X_q[5] - J[2,1]*X_q[4]) + X_q[5]*(J[1,2]*X_q[5]-$$
$$J[2,2]*X_q[4]) + X_q[6]*(J[1,3]*X_q[5] - J[2,3]*X_q[4])))/(J[1,1]*J[2,2]*J[3,3]-$$
$$J[1,1]*J[2,3]*J[3,2] - J[1,2]*J[2,1]*J[3,3] + J[1,2]*J[2,3]*J[3,1]+$$
$$J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1])$$

$$\dot{Q} = -((J[1,1]*J[3,3] - J[1,3]*J[3,1])*(c_t*d*U_q[1]^2 - c_t*d*U_q[3]^2+$$
$$X_q[4]*(J[1,1]*X_q[6] - J[3,1]*X_q[4]) + X_q[5]*(J[1,2]*X_q[6]-$$
$$J[3,2]*X_q[4]) + X_q[6]*(J[1,3]*X_q[6] - J[3,3]*X_q[4])))/(J[1,1]*J[2,2]*J[3,3]-$$
$$J[1,1]*J[2,3]*J[3,2] - J[1,2]*J[2,1]*J[3,3]+$$
$$J[1,2]*J[2,3]*J[3,1] + J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1])-$$
$$((J[2,1]*J[3,3] - J[2,3]*J[3,1])*(c_t*d*U_q[2]^2 - c_t*d*U_q[4]^2+$$
$$X_q[4]*(J[2,1]*X_q[6] - J[3,1]*X_q[5]) + X_q[5]*(J[2,2]*X_q[6]-$$
$$J[3,2]*X_q[5]) + X_q[6]*(J[2,3]*X_q[6] - J[3,3]*X_q[5])))/(J[1,1]*J[2,2]*J[3,3]-$$
$$J[1,1]*J[2,3]*J[3,2] - J[1,2]*J[2,1]*J[3,3] + J[1,2]*J[2,3]*J[3,1]+$$
$$J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1]) - ((J[1,1]*J[2,3]-$$
$$J[1,3]*J[2,1])*(-c_q*U_q[1]^2 + c_q*U_q[2]^2 - c_q*U_q[3]^2+$$
$$c_q*U_q[4]^2 + X_q[4]*(J[1,1]*X_q[5] - J[2,1]*X_q[4]) + X_q[5]*(J[1,2]*X_q[5]-$$
$$J[2,2]*X_q[4]) + X_q[6]*(J[1,3]*X_q[5] - J[2,3]*X_q[4])))/(J[1,1]*J[2,2]*J[3,3]-$$
$$J[1,1]*J[2,3]*J[3,2] - J[1,2]*J[2,1]*J[3,3] + J[1,2]*J[2,3]*J[3,1]+$$
$$J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1])$$

(A.2)

$$\dot{R} = ((J[1,1]*J[3,2] - J[1,2]*J[3,1])*(c_t*d*U_q[1]^2 - c_t*d*U_q[3]^2 + X_q[4]*(J[1,1]*X_q[6]- J[3,1]*X_q[4]) + X_q[5]*(J[1,2]*X_q[6] - J[3,2]*X_q[4]) + X_q[6]*(J[1,3]*X_q[6]- J[3,3]*X_q[4])))/(J[1,1]*J[2,2]*J[3,3] - J[1,1]*J[2,3]*J[3,2] - J[1,2]*J[2,1]*J[3,3]+ J[1,2]*J[2,3]*J[3,1] + J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1]) + ((J[2,1]*J[3,2]- J[2,2]*J[3,1])*(c_t*d*U_q[2]^2 - c_t*d*U_q[4]^2 + X_q[4]*(J[2,1]*X_q[6] - J[3,1]*X_q[5])+ X_q[5]*(J[2,2]*X_q[6] - J[3,2]*X_q[5]) + X_q[6]*(J[2,3]*X_q[6]- J[3,3]*X_q[5])))/(J[1,1]*J[2,2]*J[3,3] - J[1,1]*J[2,3]*J[3,2]- J[1,2]*J[2,1]*J[3,3] + J[1,2]*J[2,3]*J[3,1] + J[1,3]*J[2,1]*J[3,2]- J[1,3]*J[2,2]*J[3,1]) + ((J[1,1]*J[2,2] - J[1,2]*J[2,1])*(-c_q*U_q[1]^2+ c_q*U_q[2]^2 - c_q*U_q[3]^2 + c_q*U_q[4]^2 + X_q[4]*(J[1,1]*X_q[5]- J[2,1]*X_q[4]) + X_q[5]*(J[1,2]*X_q[5] - J[2,2]*X_q[4])+ X_q[6]*(J[1,3]*X_q[5] - J[2,3]*X_q[4])))/(J[1,1]*J[2,2]*J[3,3] - J[1,1]*J[2,3]*J[3,2]- J[1,2]*J[2,1]*J[3,3] + J[1,2]*J[2,3]*J[3,1] + J[1,3]*J[2,1]*J[3,2] - J[1,3]*J[2,2]*J[3,1])$$

$$\dot{x} = X_q[12]*(sin(X_q[1])*sin(X_q[3]) + cos(X_q[1])*cos(X_q[3])*sin(X_q[2]))- X_q[11]*(cos(X_q[1])*sin(X_q[3]) - cos(X_q[3])*sin(X_q[1])*sin(X_q[2]))+ X_q[10]*cos(X_q[2])*cos(X_q[3])$$

$$\dot{y} = X_q[11]*(cos(X_q[1])*cos(X_q[3]) + sin(X_q[1])*sin(X_q[2])*sin(X_q[3]))- X_q[12]*(cos(X_q[3])*sin(X_q[1]) - cos(X_q[1])*sin(X_q[2])*sin(X_q[3]))+ X_q[10]*cos(X_q[2])*sin(X_q[3])$$

$$\dot{x} = X_q[12]*cos(X_q[1])*cos(X_q[2]) - X_q[10]*sin(X_q[2]) + X_q[11]*cos(X_q[2])*sin(X_q[1])$$

$$\dot{U} = -gz*sin(X_q[2])$$

$$\dot{V} = gz*cos(X_q[2])*sin(X_q[1])$$

$$\dot{W} = (c_t*U_q[1]^2 + c_t*U_q[2]^2 + c_t*U_q[3]^2 + c_t*U_q[4]^2)/m + gz*cos(X_q[1])*cos(X_q[2]) \tag{A.3}$$

Appendix B

Navig8 End-Effector Trajectory Following

Description

This video shows a simulation of the Navig8-based UAM where Navig8 carries out a pitch motion while the manipulator follows a specified desired end-effector trajectory.

File name

Navig8.mp4

Appendix C

UAM Navigation To Object

Description

This video shows a simulation of a quadcopter-based UAM carrying out an object-grasping task within a building. In the process the UAM flies through a narrow window opening.

File name

Object-Grasp.mp4