Robust Control of Nonlinear Systems with applications to Aerial Manipulation and Self Driving Cars

by

Gowtham Garimella

A dissertation submitted to The Johns Hopkins University

in conformity with the requirements for the degree of

Doctor of Philosophy

Baltimore, Maryland

March, 2018

Abstract

This work considers the problem of planning and control of robots in an environment with obstacles and external disturbances. The safety of robots is harder to achieve when planning in such uncertain environments. We describe a robust control scheme that combines three key components: system identification, uncertainty propagation and trajectory optimization. Using this control scheme we tackle three problems. First, we develop a Nonlinear Model Predictive Controller (NMPC) for articulated rigid bodies and apply it to an aerial manipulation system to grasp an object mid-air. Next, we tackle the problem of obstacle avoidance under unknown external disturbances. We propose two approaches, the first approach using adaptive NMPC with open-loop uncertainty propagation and the second approach using Tube NMPC. After that, we introduce dynamic models which use Artificial Neural Networks (ANN) and combine them with NMPC to control a ground vehicle and an aerial manipulation system. Finally we introduce a software framework for integrating the above algorithms to perform complex tasks. The software framework provides users the ability to design systems that are robust to control and hardware failures where preventive action is taken before-hand. The framework also allows for safe testing of control and task logic in simulation before evaluating on the real robot. The software framework is applied to an aerial manipulation system to perform a package sorting task, and extensive experiments demonstrate the ability of the system to recover from failures.

In addition to robust control, we present two related control problems. The first problem pertains to designing an obstacle avoidance controller for an underactuated system that is Lyapunov stable. We extend a standard gyroscopic obstacle avoidance controller to be applicable to an underactuated system. The second problem addresses the navigation of a Unmanned Ground Vehicle (UGV) on a unstructured terrain. We propose using NMPC combined with a high fidelity physics engine to generate a reference trajectory that is dynamically feasible and accounts for unsafe areas in the terrain.

Thesis Committee

Primary Readers

Marin Kobilarov (Primary Advisor)
Assistant Professor
Department of Mechanical Engineering
Johns Hopkins University

Louis Whitcomb
Professor
Department of Mechanical Engineering
Johns Hopkins University

Joseph Moore
Senior Professional Staff
Johns Hopkins University Applied Physics Laboratory

Acknowledgments

I would like to take this opportunity to thank both personal friends and colleagues who have been instrumental in my journey as a doctoral student at Johns Hopkins University.

Firstly, I would like to acknowledge Prof. Marin Kobilarov for being a great advisor and also for introducing me to formal research in robotics. Before my Ph.D., I was interested in robotics mostly as a hobby and did not consider myself a serious roboticist. Marin introduced me to aerial robotics and emphasized the role of mathematics (as in theorems and proofs) in developing practical solutions to complicated robotics problems. I also greatly enjoyed teaching students for controls classes taught by Marin. Teaching students allowed me to develop deeper knowledge about basic concepts in research.

I also want to thank my thesis committee: Prof. Louis Whitcomb and Dr. Joseph Moore for their insightful comments about this thesis. Louis was also my academic advisor during my Masters program. I am grateful to him for guiding me on proper courses to take and general advice during my initial years at Hopkins. Joe has been a collaborator in addition to being on my thesis committee. I particularly enjoyed academic discussions I had about ensuring safety of robots in real-life.

Industrial collaboration can provide reseources that are not otherwise available in an academic setting. I would like to sincerely thank Zoox Inc for letting me use their facilities for developing algorithms and testing them on self-driving cars. I also learned a lot about how robotics algorithms are developed in an industrial setting. I also enjoyed discussing about modeling robots with several Zoox employees.

I also take this opportunity to thank my labmates Matt and Subhransu for being part of my Ph.D. journey and part of building a functional research lab out of an empty room. I was also lucky to be part of a group of great coders who can build professional codebases for robots.

On a personal note, first, I thank my parents for encouraging me to try and reach my full potential and my brother for being my early mentor in all my academic decisions. I also want to thank my hopkins friends who have been literally a family away from home and provided a lot of emotional support throughout my research journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Consider a robot such as a ground vehicle shown in Figure 1.1 navigating through a forest. To successfully navigate the forest, the robot needs an accurate model of how it will move when a specific control has been applied which is known as the robot dynamics. The robot dynamics is used to design a controller that can guide the robot along the right path while avoiding trees and unsafe areas. In practice, however, we only know the robot dynamics approximately and we have to rely on feedback from sensors such as GPS to correct the control applied to the robot to guide it back to the correct path. One of the drawbacks of pure feedback control is that there is no guarantee that the controller will be able to counter the disturbances such as external forces and model inaccuracies. Robust controllers on the other hand, account for the uncertainty in the robot dynamics and guarantee safety even under disturbances under certain assumptions about the disturbances. Traditionally, several robust control methods have been developed for linear systems such as $\mathbb{H}_\infty$ control [237], Linear Quadratic Gaussian (LQG) control [94], and Robust Model Predictive Control(MPC) [120]. The robust control schemes were later extended to nonlinear systems through methods such as Lyapunov redesign [229] and sliding mode control [211, 98]. The general drawback of feedback controllers is that they are designed to perform simple tasks such as achieving a goal state for the robot. Adding constraints on the robot and satisfying secondary criteria are hard to incorporate into feedback control. For example, we can design a controller to track a smooth trajectory for a car-like robot [43], but ensuring that the controller is satisfying control bounds such as maximum steering angle and minimizing lateral accelerations is hard, since finding the new controllers requires searching for a Lyapunov candidate that satisfies secondary criteria. Model Predictive Control (MPC) [29] has been developed for linear systems to satisfy these requirements. MPC solves the constrained optimization problem at every control step where the task that the robot has to perform is encoded as a cost function and associated constraints and the inputs to the optimization are the controls to be applied to the robot. Previous work has shown that MPC has unique solutions when the cost function is quadratic [230]. Thus given enough computational resources, we can always find the optimal control to be applied to the robot at every step.

Similar to MPC, Nonlinear Model Predictive Control (NMPC) finds optimal trajectories for nonlinear systems. NMPC solves a nonlinear optimization problem at each step. Unlike MPC, we cannot always guarantee a unique solution for NMPC [188]. There also exist additional variations of NMPC which will be explained in Chapter 2.

This work tries to formulate a robust control strategy using Nonlinear Model Predictive Control (NMPC) for general nonlinear systems such as the

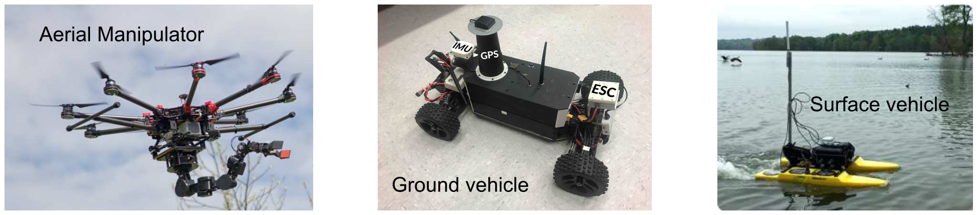


Figure 1.1: Example robotic systems for which current methods can be applied to

ones shown in Figure 1.1 and apply it to quadrotor systems and autonomous ground vehicles. There are several challenges to developing and implementing a robust NMPC system for nonlinear systems: real-time implementation, designing an accurate model of robot dynamics and tackling noise due to modeling and external disturbances. There are three main components in the proposed robust control scheme: system identification, uncertainty propagation, and trajectory optimization as explained below

1.1 System Identification

The robot dynamics is usually defined using parameters that are based on characteristics of the system. For example, the distance moved by a wheeled robot is determined by the radius of its wheels. Similarly, the acceleration of a rigid body is determined by the mass and the force applied to the body (Newton's second law). Thus, system parameters are necessary to define the relation between the control inputs and the motion of the vehicle.

System identification is the process of finding the system parameters by collecting measurements from sensors attached to the robot, and controls applied to the robot. The result of system identification is a dynamic model that can approximately describe how the robot would move based on the controls applied and the current robot state. General system identification methods are explained below.

1.1.1 Online vs Offline

In an offline method, the measurements and controls from the robot are collected first, and parameters are computed later. In contrast, online methods consume one measurement at a time and update the system parameters. Online methods can respond to changes in slow changing system parameters. The recursive least squares [72] method is an example of an online system identification method, and the least squares method is an example of an offline method.

1.1.2 Parametric vs Non-Parametric

Parametric methods assume the dynamic model of a robot is represented by a finite number of system parameters. On the other hand, non-parametric methods determine the number of system parameters based on the data [206]. In principle, a non-parametric method can fit the data exactly by using an infinite number of system parameters, but doing so will perform poorly on unseen data. The non-parametric methods are usually regularized to limit the number of parameters so that it generalizes well to unseen data.

Neural network models are by definition parametric since they have a fixed number of weights that are learnt from data. But, unlike traditional parametric methods, the structure of the model is not chosen based on apriori information known about the robot. Instead, neural networks use a biologically inspired model [63] which does not change across different systems. This allows the same network to represent different dynamic models without changing its internal structure. In this respect, the neural networks can be considered non-parametric as long as we have large enough network to cover the space of the dynamic models of interest.

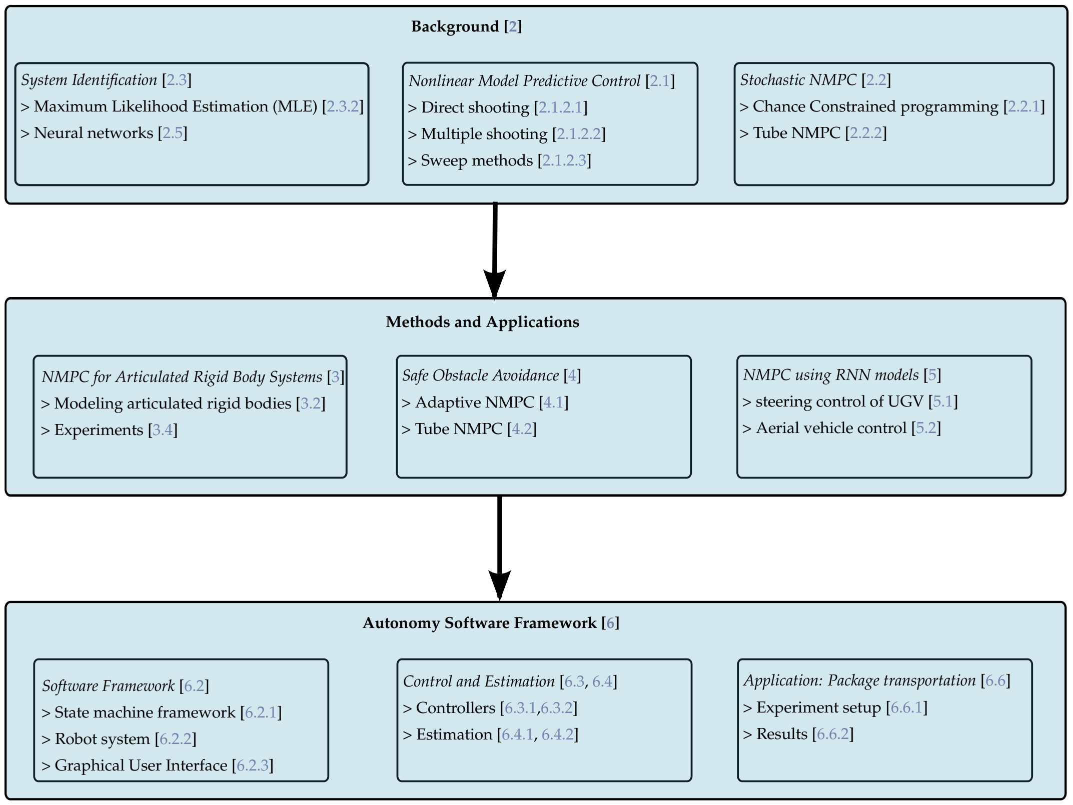
1.2 Uncertainty Propagation

The dynamic model of a robot is not perfect and thus the robot motion cannot be predicted exactly. In addition, there are external factors such as wind forces on the robot which change with time and cannot be predicted before-hand. Uncertainty propagation characterizes the distribution of the robot state based on the current state and controls applied to the robot. For example, knowing the mean and covariance of the robot state, we can move the robot in such a way that it avoids obstacles with high probability. Similarly, if we know the bounds on the external factors such as wind, we can design controllers to converge to a desired trajectory.

Uncertainty propagation characterizes the error between the robot state and desired state based on the bounds on external disturbances. The state error is used in planning robot trajectories to avoid obstacles and other unsafe areas. Uncertainty propagation can be based on assumptions that the external forces are stochastic in nature (sampled from a distribution) or just simply assumed to be bounded. In the former case, a stochastic ODE needs to be solved to find the distribution of the state [77]. In the later case, disturbance invariant sets [112] are computed. These methods will be detailed in later chapters.

1.3 Trajectory Optimization

Trajectory optimization uses an identified dynamic model to find a desired trajectory that minimizes a user-defined cost function. The cost function is based on a user's goal such as avoiding obstacles and reaching a target location. The uncertainty in the robot state is also accounted for when performing trajectory optimization. Different trajectory optimization methods and ways to solve them efficiently are explained later.



Background [2]
System Identification [2.3]
> Maximum Likelihood Estimation (MLE) [2.3.2]
> Neural networks [2.5]
Nonlinear Model Predictive Control [2.1]
> Direct shooting [2.1.2.1]
> Multiple shooting [2.1.2.2]
> Sweep methods [2.1.2.3]
Stochastic NMPC [2.2]
> Chance Constrained programming [2.2.1]
> Tube NMPC [2.2.2]
Methods and Applications
NMPC for Articulated Rigid Body Systems [3]
> Modeling articulated rigid bodies [3.2]
> Experiments [3.4]
Safe Obstacle Avoidance [4]
> Adaptive NMPC [4.1]
> Tube NMPC [4.2]
NMPC using RNN models [5]
> steering control of UGV [5.1]
> Aerial vehicle control [5.2]
Autonomy Software Framework [6]
Software Framework [6.2]
> State machine framework [6.2.1]
> Robot system [6.2.2]
> Graphical User Interface [6.2.3]
Control and Estimation [6.3, 6.4]
> Controllers [6.3.1,6.3.2]
> Estimation [6.4.1, 6.4.2]
Application: Package transportation [6.6]
> Experiment setup [6.6.1]
> Results [6.6.2]

Figure 1.2: Thesis layout

1.4 Organization

Figure 1.2 shows a flowchart of the thesis organization. The rest of the chapters are organized as follows. Chapter 2 provides the mathematical background in trajectory optimization and system identification necessary for understanding the rest of the chapters. Chapter 3 demonstrates the effectiveness of using trajectory optimization for nonlinear systems. In particular, we find optimal reference trajectories for articulated rigid bodies (aerial manipulator in our case) and track them using a low-level controller. Continuing, Chapter 4 develops NMPC techniques for safe obstacle avoidance under model uncertainty and external disturbanes using adaptive NMPC and Tube NMPC. Chapter 5 overcomes the necessity to design dynamic models for nonlinear systems by combining NMPC with Recurrent Neural Networks (RNN) to control general nonlinear systems. Finally, a general software framework for task-level control is described in 6 and is applied to a package transportation task. Concluding remarks are presented in Chapter 8. Chapter 7 develops two related control techniques: gyroscopic obstacle avoidance controller for underactuated systems and NMPC using physics based models.

Chapter 2

Background

This chapter describes different trajectory optimization methods to control a robot in stochastic and deterministic settings and explains basic methods to estimate the robot dynamics and find the covariance of a nonlinear function. These methods are applied to quadrotor systems and autonomous vehicles to achieve robust control in later chapters.

2.1 Nonlinear Model Predictive Control (NMPC)

The goal of NMPC is to compute a dynamically feasible trajectory that minimizes a user defined cost. The cost encodes the task level specification such as moving the robot to a target pose. There are two main components of NMPC: the dynamic model and trajectory optimization. The dynamic model of a robot is given by an Ordinary Differential Equation (ODE) as

$$\dot{x} = f(t, x, u), \tag{2.1}$$

where the state of the system is given by $x \in T$, controls given by $u \in \mathbb{R}$, and time given by $t \in \mathbb{R}^+$. The dynamics can be discretized as

$$x_{i+1} = f(t_i, x_i, u_i), \tag{2.2}$$

where the subscript i indicates the time segment along the trajectory. It is also assumed that the control u_i is constant during the time interval $[t_i, t_{i+1}]$. The discretized dynamics can be derived from the continous dynamics as

$$f(t_i, x_i, u_i) = x_i + \int_{t_i}^{t_{i+1}} f(t, x(t), u_i)dt, \tag{2.3}$$

where the ODE is integrated over a small time assuming constant control. When the state is on a manifold ($x \in T$), the addition operator in (2.3) can be thought of as moving on the manifold and not a simple addition. For a small discretization step $h_i = t_{i+1} - t_i$, the discretized dynamics can be written as $f(t_i, x_i, u_i) = x_i + h_i f(t_i, x_i, u_i)$. The discretized dynamics is then integrated to predict the state of system in the future, i.e., given the initial state x_0 at time t_0, and the future times $t_{1:N}$ and controls applied $u_{0:N-1}$, we can find the states of the system $x_{1:N}$ along the trajectory. Using this predicted trajectory, we can perform trajectory optimization to minimize the cost function. In this work, we are only explaining the discrete NMPC formulation and not the continous version using calculus of variations which can be found in Bertsekas [19].

2.1.1 Trajectory Optimization

The goal of trajectory optimization is to find a set of controls $u_{1:N}$ which minimize a user-defined cost on the trajectory. The cost function encodes the task specification. For example, if we want to move the robot to a desired state x_d at time t_f, the cost function can be specified as the distance between the trajectory state at t_f and the desired state:

$$J(x_{0:N}, u_{0:N-1}) = \phi(x(t_f), x_d)^T Q_f \phi(x(t_f), x_d), \tag{2.4}$$

where $\phi(\cdot, \cdot) \in \mathbb{R}$ is the displacement vector between x_d, $x(t_f)$ and $Q_f \in \mathbb{R}^{n \times n}$ is the gain matrix. In general, the trajectory cost function for many problems can be written as

$$J(x_{0:N}, u_{0:N-1}) = L_N(x_N, x_d) + \sum_{i=0}^{N-1} L_i(x_i, u_i), \tag{2.5}$$

where x_i denotes the state at time t_i. The cost L_N denotes the terminal cost that moves the end of the trajectory towards a goal state x_d, and the stagewise cost L_i minimizes the control effort and velocities along the trajectory.

The discrete NMPC problem performs a nonlinear optimization at every control step. The nonlinear optimization can be summarized as

$$\min_{u_{1:N}} J(x_{0:N}, u_{0:N-1}), \tag{2.6}$$

$$\text{s.t } x_{i+1} = f(t_i, x_i, u_i). \tag{2.7}$$

The result of the optimization is a sequence of controls which will ensure the robot follows the optimal trajectory in open-loop under perfect conditions. The first control u_0 is applied to the robot and the optimization procedure is repeated. In practice, if the optimization frequency is not fast enough, a few of the control samples from the previous optimization are applied in open-loop using a time based lookup before the procedure is repeated. The controls at the next step are initialized by cycling the controls from the previous optimization to reduce the number of iterations until convergence. The NMPC procedure is summarized in 1.

Algorithm 1 NMPC Procedure

```
Given $x_d$
Converge $\leftarrow$ False
Initialize controls to steady state
while not Converge do
    Initialize controls $u^*$ from past iterations
    $u^*_{1:N} = \min J$
    Lookup control based on time/state
    Send control $u^*_j$ to robot
end while
```

2.1.2 Methods for Trajectory Optimization

The trajectory optimization shown in (2.7) is a nonlinear optimization which can be solved using first order methods such as gradient descent or second order methods such as the Gauss-Newton method. Directly solving the optimization problem without using its internal structure results in an optimization problem with very high dimension which cannot be solved in real time. Below, we explain a few methods that can tackle the trajectory optimization effectively. There also exist a few methods to solve optimization using Euler-Lagrange methods that can be found in Bertsekas [19] and are not discussed in this work.

2.1.2.1 Direct Shooting

In direct shooting, the control trajectory is parametrized using a condensed vector $\xi \in \mathbb{R}^{N_\xi}$ i.e. the control at time t_i is given by $u_i = \psi(t_i, \xi)$. There are several parametrizations possible for the control vector. For example, we could use a spline parametrization such as $u(t_i) = \sum_{k=1}^{N_p} B_k(t_i)\xi_k$, where the controls are parametrized using the knot vector $\xi = [\xi_1, \xi_2, \cdots, \xi_{N_p}]$. The control trajectory is converted to the state trajectory by integrating the dynamics. Thus, the trajectory cost is now parametrized using fewer parameters i.e. the knot vector. The optimization can then be solved using any of the regular optimization methods such as Gradient Descent [197], Coordinate Descent [66] with the inputs given by the control parameters ξ_i. The direct shooting optimization can be formulated as

$$\min_{\xi} J(x_{0:N}, u_{0:N-1}), \tag{2.8}$$

$$\text{s.t } u_i = \psi(t_i, \xi), \tag{2.9}$$

$$x_{i+1} = f(t_i, x_i, u_i). \tag{2.10}$$

Cons: The trajectory optimization problem (2.7) is inherently sparse since the controls at stage i only affect the states $x_{i+1:N}$, and therefore the gradient of any of the states x_i with controls $u_{i+1:N}$ is zero. Direct shooting destroys the sparsity by compressing the controls using a parametrization ξ. This can sometimes lead to slower computational time. Further, by restricting the controls to a specific parametric class, the optimality of the trajectory optimization can be limited. To understand this better, let us consider a linear dynamics i.e. $x_{i+1} = A_i x_i + B_i u_i$. Further, let us parametrize our controls to be linear in time i.e. $u_i = \xi_{i_1} t_i + \xi_{i_2}$ (ξ_{i_1, i_2} are slope and intercept). If we choose a quadratic cost, the optimal controls are obtained by linear feedback controller where the controls are linear in the state and not in time. Thus, trajectories obtained by direct shooting is optimal only within the control parametrization class selected. In practice, however, the parametrization is tweaked by the user to have sufficient diversity to achieve optimal trajectories while also minimizing the number of inputs to the optimization. Direct shooting works only with small time horizons since a small change in model parameters leads to a large change in the trajectory. Direct shooting is also a local optimization method which is susceptible to local minima often. Finally, the algorithmic complexity of direct shooting scales poorly with the length of parameter vector.

2.1.2.2 Direct Multiple Shooting

In this method, the trajectory to be optimized is broken into k components. The controls along each component are assumed to be parametrized using ξ_j. The controls given by ξ_j propagate the state along jth segment. The continuity of the state between successive components is added as a constraint to the optimizer. The complete trajectory optimization is formulated as a nonlinear optimization:

$$\min_{\xi_{1:k}, x_{1:k}} \sum_{j=1}^{k} J_j, \tag{2.11}$$

$$x'_{j+1} = \psi(x_j, \xi_j), \tag{2.12}$$

$$x_{j+1} = x'_{j+1}. \tag{2.13}$$

The nonlinear optimization minimizes the sum of trajectory costs of all components subject to the constraint that the components are connected together. This type of optimization allows for sparsity to be enforced across components when solving the optimization. Since the trajectory is composed of multiple components, the controls in each parameter only affect that component and the neighboring components.

Multiple shooting enforces connectivity between different components as an equality constraint in optimization. This allows user to initialize the states at the end of each components $\bar{x}'_j$ based on initial guess from a higher level planner. The optimizer then tries to find the controls to enforce the connectivity. Multiple shooting is less susceptible to local minima by allowing the user to guess the initial states closer to true state and thereby works with longer horizon than direct shooting.

Cons: The size of the optimization problem is increased dramatically since we have to optimize over the parametrization of all the components. Incorporating state constraints into the optimization requires a large number of components since constraints can only be enforced on the ends of each components.

2.1.2.3 Sweep Methods

Sweep methods are similar to direct shooting method but use the sparsity encoded in the trajectory optimization problem. Here, we introduce Differential Dynamic Programming (DDP) which approximately solves the Bellman equation:

$$V_i(x_i) = \min_{u_i} Q_i(x_i, u_i) = \min_{u_i} \left(L_i(x_i, u_i) + V_{i+1}(f(x_i, u_i)) \right), \tag{2.14}$$

where $V(x_i)$ is the value function which denotes the minimal cost that can be achieved starting at x_i and $Q_i(x_i, u_i)$ is Q value which is equal to the sum of cost accrued at stage i given by $L_i(x_i, u_i)$ and the value function evaluated at the next state $x_{i+1} = f(x_i, u_i)$.

There are two main components in DDP: a backward pass and a forward pass. In the backward pass, the direction in which to move the control is obtained by solving a second order approximation of the Bellman equation at each stage. The second order approximation of the Bellman equation can be written as

$$\Delta Q_i \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta x_i \\ \delta u_i \end{bmatrix}^T \begin{bmatrix} 0 & \nabla_x Q_i^T & \nabla_u Q_i^T \\ \nabla_x Q_i & \nabla_x^2 Q_i & \nabla_{xu} Q_i \\ \nabla_u Q_i & \nabla_{ux} Q_i & \nabla_u^2 Q_i \end{bmatrix} \begin{bmatrix} 1 \\ \delta x_i \\ \delta u_i \end{bmatrix} \tag{2.15}$$

The optimal control perturbation δu_i^* minimizing Q_i can be obtained as

$$\delta u_i^* = K_i \delta x_i + \alpha_i k_i, \tag{2.16}$$

where $K_i = -\nabla_u^2 Q_i^{-1} \nabla_{ux} Q_i$ and $k_i = -\nabla_u^2 Q_i^{-1} \nabla_u Q_i$. Assuming the dynamics is linearized as $\delta x_{i+1} = A_i \delta x_i + B_i \delta u_i$, the backward pass algorithm is summarized as shown in Algorithm 2

Algorithm 2 DDP-Backward

```
$V_x = \nabla_x L_N$
$V_{xx} = \nabla_{xx} L_N$
for $k = N - 1 \to 0$ do
    $Q_x = \nabla_x L_i + A_i^T V_x$
    $Q_u = \nabla_u L_i + B_i^T V_x$
    $Q_{xx} = \nabla_{xx} L_i + A_i^T V_{xx} A_i$
    $Q_{uu} = \nabla_{uu} L_i + B_i^T V_{xx} B_i$
    $Q_{ux} = \nabla_{ux} L_i + B_i^T V_{xx} A_i$
    $V_x = Q_x + K_i^T Q_u$
    $V_{xx} = Q_{xx} + K_i^T Q_{ux}$
end for
```

Algorithm 3 DDP-Forward

```
$\delta x_0 = 0\ V'_0 = 0$
while do$V'_N > V_N$ or Termination
    $\alpha =$ Armiho()
    for do$k = 0 \to N - 1$
        $u'_i = u_i + \alpha k_i + K_i \delta x_i$
        $x'_{i+1} = f_i(x_i, u'_i)$
        $V'_{i+1} = V'_i + L_i(x'_i, u'_i)$
    end for
    $V'_N = V'_{N-1} + L_N(x'_N)$
end while
```

Once the direction for control perturbation is obtained, a line search is applied during the forward pass to find a step size that minimizes the trajectory cost. The pseudo code for the forward pass is shown in Algorithm 3. Finally, the backward and forward pass are repeated until convergence. In the backward pass, a regularization term is usually added to ensure that the Hessian of Q value is positive definite.

Pros: The computational complexity of the method is linear in the number of trajectory steps as opposed to cubic for a naive usage of Gauss-Newton method. DDP also converges faster than a gradient descent method since it uses Hessian information from the Q value during the backward pass.

Cons: It is harder to incorporate state and control constraints into sweep methods. Usually sweep methods work best when there are only box constraints on the controls and no state constraints. DDP is also a local optimization method and is also susceptible to local minima.

2.2 Stochastic NMPC

Stochastic systems are robotic systems for which we cannot exactly predict the state of the system based on the current state and controls. Instead, the dynamics of the system evolves using a stochastic ODE where the state of the system at each point is a random variable. For example, the discrete version of the stochastic ODE can be written as

$$x_{i+1} = f(x_i, u_i, w_i), \tag{2.17}$$

where x_i is the state random variable at stage i, u_i is the control, and w_i is the perturbation or noise introduced at stage i. The noise can correspond to either forces applied by unknown agents or our inability to model the dynamic system accurately. Since we cannot predict the state of the system exactly, we have to work with the distribution of the trajectories based on applied controls.

The goal of stochastic NMPC is to minimize the expected cost of the trajectory where the expectation is taken over the perturbations and the initial state distribution. Differential Dynamic Programmming (DDP) described in section 2.1.2.3 can be extended to solve stochastic NMPC [219]. Adding deterministic state constraints to stochastic NMPC is not possible since the state at each stage along the trajectory is sampled from a distribution.

Pros: Stochastic NMPC handles uncertainty in the model by minimizing the expected trajectory cost.

Cons: Finding an exact gradient of the expected cost is non-trivial for general stochastic nonlinear systems. Usually Monte Carlo simulations are used to find the gradient of the expected cost but they are computationally inefficient and are not practical for higher dimensional systems.

2.2.1 Chance Constrained Programming

Chance constrained programming is a method that deals with solving stochastic NMPC with constraints [34]. It replaces deterministic constraints with probabilistic constraints as shown in (2.19).

$$\min_{u_{1:N}} E_{x_0,w_{0:N-1}}[J], \tag{2.18}$$

$$P[\psi(x_i) \leq 0] \geq 1 - \delta, \tag{2.19}$$

where δ determines how unlikely that the constraint will be violated. Chance constrained programming has been initially limited to linear systems [205]. Recently chance constrained programming has been extended to nonlinear systems [54]. One of the major drawbacks of the existing methods is that they are computationally expensive and cannot be run in real-time.

2.2.2 Tube NMPC

Instead of dealing with stochastic systems, there are a set of methods that do not assume the disturbances/perturbations are sampled from some distribution. Tube-NMPC belongs to this class of methods. Tube-NMPC assumes that the disturbances are bounded by a user-defined constant and thereby does not assume anything about the distribution of the disturbances. The goal of Tube-NMPC is to satisfy constraints for the dynamic system such as $\psi(x) \leq 0$ (x is the state of the system) even under the influence of the disturbances w assuming disturbances are bounded $\|w\| \leq \epsilon$. The trajectory optimization for Tube-NMPC can be formulated as

$$\min_{u_{1:N}} J(x_{0:N}, u_{0:N-1}), \tag{2.20}$$

$$\text{s.t } \psi(x_i) \leq 0, \quad \forall \|w_i\| \leq \epsilon_i, \tag{2.21}$$

$$x_{i+1} = f(x_i, u_i, w_i). \tag{2.22}$$

To solve such an optimization problem, we need to find the region in which the state x_i will lie in under the influence of disturbances. The constraints are then satisfied for every point in the region. The disturbance invariant region (or simply invariant region) at stage $i+1$ is therefore defined as

$$P_{i+1} = \{x_{i+1} : x_{i+1} = f(x_i, u_i, w_i), \|w_i\| \leq \epsilon_i, x_i \in P_i\}, \tag{2.23}$$

where the control u_i is assumed to be either a controller ($u_i = g(x_i)$) or a deterministic value in which case the size of the invariant region cannot be regulated explicitly.

To solve the optimization in (2.20-2.22), the constraints are modified to be applied on the invariant region, i.e. $\bar{\psi}(P_{i+1}) \leq 0$. This allows the system to navigate obstacles and satisfy actuator constraints even when subject to bounded disturbances. One of the main tasks in a Tube NMPC is to find the invariant regions for a general nonlinear system. Majumdar and Tedrake [142] used Sum Of Squares (SOS) programming techniques to compute the invariant region for polynomial systems offline. Recent methods extended the invariant funnel computation to nonlinear systems with a known Lipschitz constant [232]. In Chapter 4.2, we propose a novel method to find the invariant region for general nonlinear systems and formulate the NMPC using the invariant region dynamics.

Cons: Tube NMPC often produces trajectories that are overly conservative since we only assume the disturbances are bounded unlike stochastic NMPC where we assume the disturbances are obtained from a distribution. Tube NMPC is also computationally expensive as compared to deterministic NMPC methods which do not consider model uncertainty.

2.3 System Identification

The NMPC optimization methods described so far relies on our ability to predict the state of a robot system at a future time given its current state and the controls applied to it, i.e. we assume we know the dynamics equation of the robot $x_{i+1} = f(x_i, u_i)$. Usually this dynamic equation also consists of parameters that are assumed to be constant and describe the robotic system. For example, the dynamics of a simple pendulum consists of the length of the rope and the mass of the bob. The goal of this section is to identify the parameters based on a given set of measurements from different sensors.

2.3.1 Least Squares Estimation

One of the fundamental methods to perform system identification is using least squares estimation. Least squares estimation minimizes a least squares cost on the error between predicted and observed measurements ($\eta, \overline{\eta} \in \mathbb{R}^{n_\eta}$). In the case of a linear measurement model, the predictions are given by a linear relationship: $\overline{\eta} = \xi\psi$, where $\xi \in \mathbb{R}^{n_\eta \times n_\psi}$ represents the optimization parameters and $\psi \in \mathbb{R}^{n_\psi}$ represents the inputs to the system. The least squares optimization for n measurements $\eta_{1:N}$ and inputs $\psi_{1:N}$ can be stated as:

$$\min_{\xi} \sum_{i=1}^{N} \|\eta_i - \xi\psi_i\|_2^2. \tag{2.24}$$

Gauss-Markov theorem states that the least squares method finds an unbiased estimate of the parameters ξ under certain conditions regarding the noise added to the measurements [4]. The variance of the estimator can be improved by knowing the distribution characteristics of the measurements.

If the measurement model is nonlinear for example $\overline{\eta} = f(\xi, \psi)$, then non-linear least squares method can be employed to find the optimal parameters ξ [71]. Unlike, least squares, we cannot guarantee a global miniminum for the cost function. Least squares method is implemented in practice using a batch method where the measurements η are collected in small quantities (batch) and the optimal parameters ξ are propagated from batch to batch [196].

2.3.2 Maximum Likelihood Estimation

The goal of maximum likelihood estimation (MLE) is to estimate the parameters of a density function of a random variable based on samples obtained from it's distribution. Let us assume that $X = \{X_1, X_2, \cdots, X_n\}$ be random variables with a joint probability density function (p.d.f) $f(x|\theta)$. The parameter vector θ is unknown and lives in some open set $\theta \in \Sigma$. The parameter vector induces a family of joint distributions on the random vector X. The goal of Maximum Likelihood estimation (MLE) is to find an estimate for the true parameters θ^*, given a sample $x = \{x_1, x_2, \cdots, x_n\}$ of the random vector X. The p.d.f evaluated at a given sample is known as the likelihood function. The principle of MLE is to choose the value of θ for which the likelihood function is maximized:

$$\hat{\theta}(x) = \arg\max_{\theta \in \Sigma} f(x|\theta) \tag{2.25}$$

By choosing such a θ, the probability of obtaining the observed sample given θ becomes high. MLE obtains the best estimate for θ which maximizes the probability of obtaining the observed sample. If a prior distribution on θ is known, then the MLE estimation should be modified to multiply the conditional density with the prior density on θ.

MLE is a consistent estimator (under certain conditions on the $f(x|\theta)$) i.e. the distribution of the estimate converges to θ^* in probability [45]. The estimate $\hat{\theta}_n$ of MLE is sampled from a function of the random vector **X**, i.e. for every sample x of random vector **X**, MLE provides an estimate $\hat{\theta}_n(x)$).

The convergence in probability implies that $\lim_{n\to\infty} P\left[\|\hat{\theta}_n - \theta^*\| \leq \epsilon\right] = 1$ for any $\epsilon > 0$. Hence MLE estimator gets closer to the actual parameters as the number of samples increase under certain conditions on the estimator distribution. Note: Although MLE estimates gets arbitrarily closer to the actual θ^*, they may **not** be unbiased ($E_X(\hat{\theta}_n(x)) \neq \theta^*$).

2.3.2.1 Log Likelihood Function

In many cases, it is advantageous to work with the log of the likelihood function. **log** is an increasing function and is concave. Thus log of a likelihood function achieves the maximum at the same θ as likelihood function. In addition log-likelihood functions separate the joint density function into a sum of density functions for an i.i.d sequence of random variables $f_X(x) = \Pi_i f_{X_i}(x_i|\theta); log f_X(x) = \sum_i log f_{X_i}(x_i|\theta)$. In such a case, taking derivatives of sums becomes easier and is usually numerically more stable than the original exponential function.

2.3.2.2 Information Inequality

This inequality defines the minimum variance possible for an arbitrary estimator of parameter θ^* from i.i.d samples x_i sampled from p.d.f $f(x_o|\theta)$ [20]. In particular it states that for an unbiased estimator, the variance of the estimate is lower bounded: $\text{Var}_X(\hat{\theta}_n(x)) \geq \frac{1}{n} I^{-1}(\theta)$. The matrix $I(\theta)$ is known as Fisher information matrix and is defined as $I_{ij}(\theta) = E[\frac{\partial}{\partial \theta_i} log(f(x_o|\theta) \frac{\partial}{\partial \theta_j} log(f(x_o|\theta)]$. An efficient estimator is one which achieves the minimum variance stated above. This type of estimator not only produces estimates close to θ^*, but also the spread of the estimates about θ^* will be smallest among all possible unbiased estimators. MLE is an efficient estimator under few regularity conditions [45].

2.3.2.3 Asymptotic Distribution of MLE

Assuming a few regularity conditions on the density function $f(x|\theta)$ as explained in Hogg and Craig [78], the distribution of the scaled error between estimated parameter and true parameter converges in distribution to a standard normal distribution: $\lim_{n\to\infty} \sqrt{nI(\theta)}(\hat{\theta}_n - \theta^*) \to N(0, I)$(matrix square root). The distributional convergence is true even for biased MLE estimators under certain conditions. For example, the MLE estimator for a standard deviation(σ) of a normal distribution is biased but still satisfies the above asymptotic normality.

2.3.3 Moving Horizon Estimator

The moving horizon estimator applies MLE to a series of sensor measurements to find the mean and covariance of the parameters of the dynamic system [7]. Assuming we have a continous series of sensor measurements $z_{1:N}$ and the initial state x_0 and controls $u_{0:N-1}$, we want find to the parameters p that satisfy

$$x_{i+1} = f(x_i, u_i, p), \tag{2.26}$$

$$z_i = h(x_i). \tag{2.27}$$

Usually, we assume that measurements are corrupted by gaussian noise with zero mean and covariance Σ_z, and therefore we cannot find true parameters that satisfy (2.27). Moving horizon estimation minimizes the log likelihood cost for the parameters given a set of sensor measurements as

$$J = \frac{1}{2}\left[\sum_{i=1}^{N}(z_i - \bar{z}_i)^T\Sigma_z^{-1}(z_i - \bar{z}_i) + (p - \bar{p})^T\Sigma_p^{-1}(p - \bar{p})\right], \tag{2.28}$$

$$\text{s.t} \quad \bar{z}_i = h(x_i), \tag{2.29}$$

$$x_{i+1} = f(x_i, u_i, p). \tag{2.30}$$

where $\bar{p}$ is a prior on the parameters and Σ_p is the covariance of the prior and Σ_z is the covariance of the sensor measurements. The mean and covariance of the parameters are obtained by minimizing the cost function $p^* = \arg\min_p J(\bar{z}_{1:N}, \bar{p}, x_0)$. The covariance for the parameters is approximated based on the Fischer information matrix of the sensor measurements:

$$J_p^* = \partial J_p\big|_{p=p^*}, \tag{2.31}$$

$$\Sigma_p \approx \left(J_p^* {J_p^*}^T\right)^{-1}, \tag{2.32}$$

where n_p is the dimension of the parameter vector p.

The prior on the parameters $\bar{p}$ and the covariance Σ_p are updated based on the optimized mean and covariance obtained from the optimization. The moving horizon estimator is used in Chapter 4.1 to find the parameters of a quadrotor model along with external disturbances such as body forces before applying NMPC on the learned model.

2.4 Unscented Transform

The unscented transform [228] is an approximate method to compute the mean and covariance of a nonlinear transformation of a random variable. Let us assume that we are given a random variable x with mean μ_x and covariance Σ_x. Further, let us assume we obtained a random variable by transforming x as $y = g(x)$. The goal of unscented transform is to compute the mean and covariance of the random variable y by selectively sampling points from x. The unscented transform first selects $2n + 1$ "sigma points" (n is the dimension of the random variable x) which are points perturbed along the columns of the square root of the covariance matrix. The sigma points $\xi_{0:2n}$ can be mathematically written as

$$\xi_0 = \mu_x, \tag{2.33}$$

$$\xi_i = \mu_x + \left(\sqrt{(n+\lambda)\Sigma_x}\right)_i, \quad i \in \{1, \cdots, n\}, \tag{2.34}$$

$$\xi_i = \mu_x - \left(\sqrt{(n+\lambda)\Sigma_x}\right)_i, \quad i \in \{n+1, \cdots, 2n\}. \tag{2.35}$$

Once the sigma points are defined, we define the mean and covariance of y as the weighted sum of propagated sigma points:

$$\mu_y = \sum_{i=0}^{2n} W_i g(\xi_i), \tag{2.36}$$

$$\Sigma_y = \sum_{i=0}^{2n} \hat{W}_i \left(g(\xi_i) - \mu_y\right) \left(g(\xi_i) - \mu_y\right)^T, \tag{2.37}$$

where,

$$W_0 = \lambda/(n+\lambda), \tag{2.38}$$

$$\hat{W}_0 = \lambda/(n+\lambda) + (1 - \alpha^2 + \beta), \tag{2.39}$$

$$\hat{W}_i = W_i = 1/\left[2(n+\lambda)\right], \;\; i \in \{1, \cdots, 2n\}. \tag{2.40}$$

The parameters λ, α, β can be optimized further based on the type of distribution of x. In Chapter 4.1, the unscented transform is used to solve a chance constrained program for a quadrotor to avoid obstacles with high probability.

2.5 Neural Networks

Neural networks are a type of function approximators which emulate roughly the biological neurons [86]. The networks are made of a single building block called the perceptron. Each perceptron consists of an affine function followed by a nonlinear activation function as shown in Figure 2.1. Different types of activation functions such as sigmoid, *tanh* and rectified linear units have been used in literature. By chaining several perceptrons together, the output of the network can approximate more and more complex functions. A single layer of a neural network can be mathematically stated as

$$y = \psi(Wx + b), \tag{2.41}$$

where $x \in \mathbb{R}^n$ is the input to the layer, $\psi(\cdot)$ is the activation function, $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are the weights and biases of the network and $y \in \mathbb{R}^m$ is the output of the neural network layer.

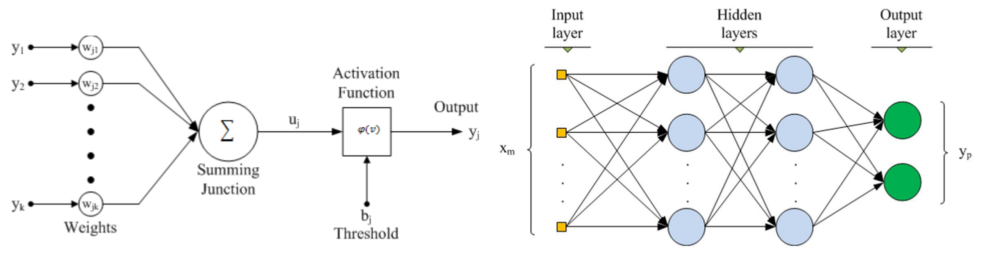


Figure 2.1: Schematic of a perceptron [227] and Neural network with multiple perceptrons connected together [111]

The basic network explained above is called a fully connected network. This type of network does not use the full structure of the problem and thus has a limited capacity to approximate complex functions. Neural networks that use the symmetry in the problem such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) have been developed for specific applications such as object detection and time-series prediction where exploiting the symmetry in the problem is necessary.

2.5.1 Training a fully connected network for supervised tasks

The goal of a supervised learning task is to find a network that can best predict the samples $\{y_{1:N}, x_{1:N}\}$. Mathematically, assuming the network is represented as the function $\bar{y} = f(x, p)$, the goal of the training is to find the parameters of the network p^* that minimize the error between predicted output $\bar{y}_{1:N}$ and the measured output $y_{1:N}$. The training problem can be formulated as

$$p^* = \arg\min_{p} L(f(x_{1:N}, p), y_{1:N}), \tag{2.42}$$

where the cost function $L(\cdot, \cdot)$ measures the error between predicted and observed measurements. For a regression problem, where the outputs $y \in \mathbb{R}^m$, the cost function is usually selected as a squared euclidean norm. This optimization problem is usually solved using a variation of stochastic gradient descent known as minibatch gradient descent [197]. In the simplest form of the algorithm, a small batch of M samples is used to find the gradient of the cost function. The parameters are then perturbed in the negative of the gradient direction with a step size that is annealed with the number of iterations. The gradient of the cost function with respect to the parameters can be computed using a specialized automatic differentiation technique known as back propagation since the cost function and structure of the network, i.e. the activation function and the connections between the layers are known beforehand.

2.5.2 Applications

Neural networks are used in a wide-variety of applications such as hand-writing and face detection from images [173], teaching computers how to play against human opponents in a game of Go [209], how to drive a car [21] and predicting stock markets [236]. A lot of research effort has also been devoted to understanding why the networks are such good function approximators [207]. One of the limitations to the traditional neural networks are that it is hard to know when the network fails. Recent research has also been focused on generating explainable neural networks that can provide some form of explanation for the output they are generating [67].

2.5.3 Recurrent Neural Networks (RNN)

Recurrent neural networks are used to predict time-series data. These networks share the same perceptron across different time steps as shown in Figure 2.2. There are different types of recurrent networks such as LSTM and GRU networks which have different types of prediction units shared across time steps [204]. By sharing the same prediction unit across time steps, a small network can learn complex time series functions by composing over time.

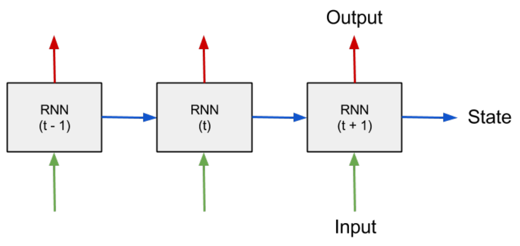


Figure 2.2: A schematic of recurrent neural network as shown in Hallstrom [70]. The weights of a prediction unit are shared across the time-steps.

To learn the weights of a RNN, a variation of back-propgation known as truncated back propagation is used to learn the gradient. Under this approach, several segments of truncated time-series samples are collected and mini-batch gradient descent is applied to the collection of samples. By truncating the time-series data, we ignore the gradients of the samples beyond the truncated time on the weights.

Chapter 3

Model Predictive Control for Articulated Rigid Body Systems

3.1 Introduction

This chapter verifies the effect of using optimal reference trajectories on general nonlinear systems. In particular, we focus on enabling agile pick-and-place capabilities for aerial vehicles equipped with manipulators through trajectory optimization. Aerial manipulation using vertical take-off and landing (VTOL) vehicles is a relatively new research area with a potential for various novel applications such as coordinated assembly, construction, and repair of structures at high altitudes, or operating in difficult-to-access, remote, or hazardous locations to e.g. install sensors or obtain samples. Autonomous control of such system is challenging primarily due to disturbances from interactions with the environment, due to additional dynamics caused by a moving manipulator, and due to difficulties associated with dexterous manipulation.

Initial work related to aerial manipulation included slung load transportation with helicopters [57, 148], grasping with novel adaptive end-effectors [180, 178], construction using teams of quadcopters [131], or pole balancing tasks [73]. More recently, there has been a focus on autonomous construction and environment interaction, with initial demonstrations in laboratory settings. The Aerial Robotics Cooperative Assembly System (ARCAS) project [13, 91, 74], Mobile Manipulating Unmanned Aerial Vehicle project [169, 117, 119] and Airobots project [3, 223] have demonstrated complex manipulation and assembly tasks using multiple degrees of freedom manipulators. Other important developed capabilities include telemanipulation [152, 89] or avian-inspired agile grasping [221]. In addition to control-related challenges, accurate pose estimation of objects is of central importance and has been considered through image-based visual servoing [220] and marker-based pose computation [13, 73]. Real-time recognition and aerial manipulation of arbitrary unengineered objects in natural settings remains largely an open problem.

Control strategies for aerial manipulation can be divided into *coupled* which consider the full multi-body system model [132, 105, 168], and *decoupled* based on separate controllers for the base body and manipulator [198]. The key difference is that the decoupled approach treats external forces from the arm or environment as disturbances to be compensated by the vehicle.

In this chapter, we propose an optimal control algorithm for generating reference trajectories to pick an object using aerial vehicle. Experimental verification has been performed using a minimalist low-cost system based on a two-degree of freedom manipulator with a simple gripper. The task is made challenging by using a monocular camera to recognize and track the target object. To facilitate recognition, objects are engineered with LED markers

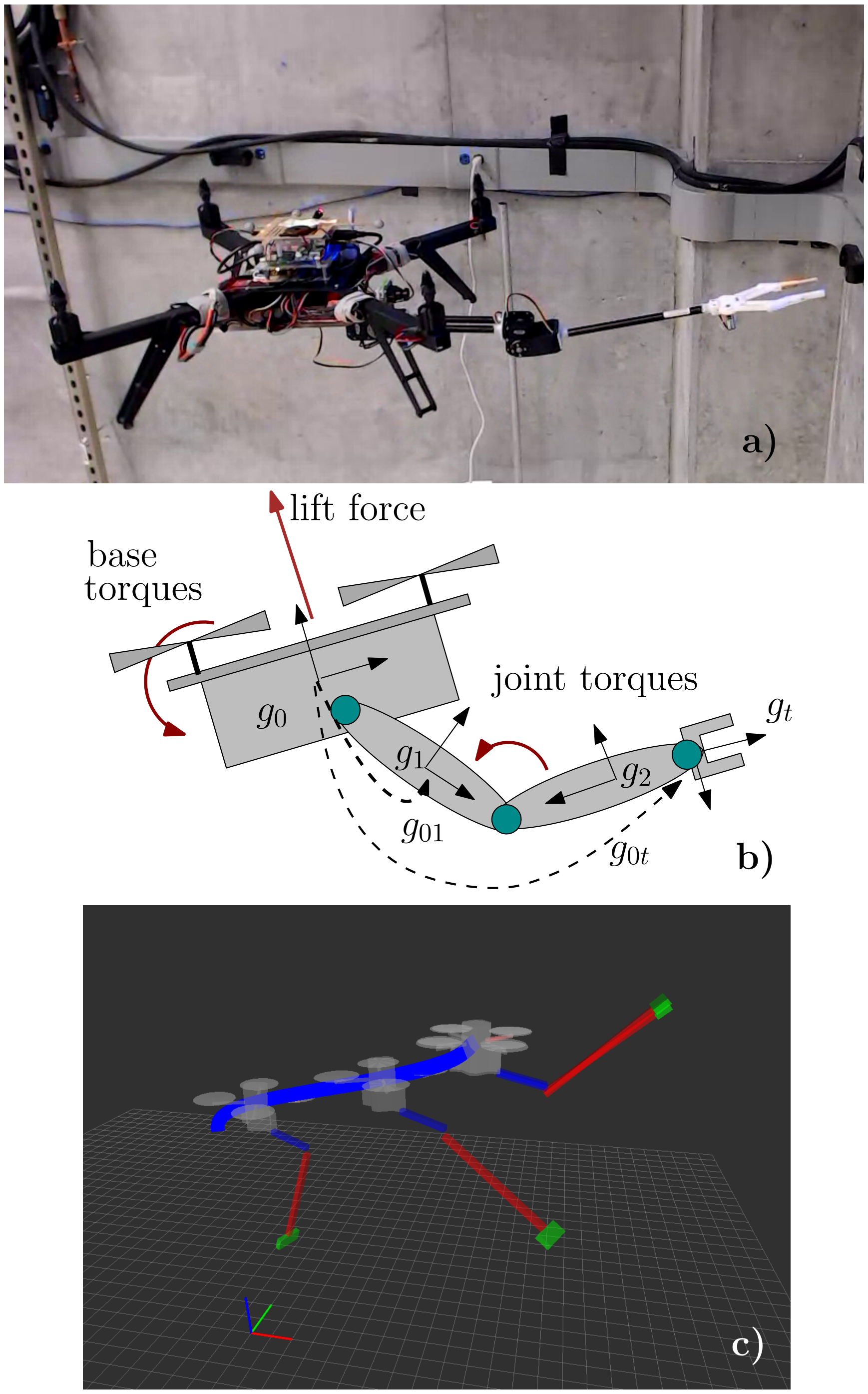


Figure 3.1: a) a prototype quadrotor with manipulator, b) schematic model, c) a computed optimal trajectory for a quadrotor platform with a manipulator attached.

correspoding to known features. A detailed nonlinear model is employed by the optimal control framework to capture the interaction between the arm and quadcopter. Currently due to computational limitations, trajectory optimization operates at 10Hz and is not used for *real-time control*. Hence a high frequency nonlinear controller is coupled with the optimal control framework to track the reference trajectories.

The chapter is organized as follows. The dynamical multi-body system modeling and numerical optimal control approach are described in section 3.2 and section 3.3, respectively. Then we proceed to describe the experiments conducted to validate the optimal controller in section 3.4. Finally we provide the results of the experiments conducted and discuss future work in section 3.5.

3.2 System Modeling for articulated rigid bodies

The aerial robot is modeled as a free-flying multi-body system consisting of $n + 1$ interconnected rigid bodies arranged in a tree structure. The configuration of body #i is denoted by $g_i \in SE(3)$ and defined as

$$g_i = \begin{pmatrix} R_i & p_i \\ 0 & 1 \end{pmatrix}, \quad g_i^{-1} = \begin{pmatrix} R_i^T & -R_i^T p_i \\ 0 & 1 \end{pmatrix}.$$

where $p_i \in \mathbb{R}^3$ denotes the position of its center of mass and and $R_i \in \mathrm{SO}(3)$ denotes its orientation. Its body-fixed angular and linear velocities are denoted by $\omega_i \in \mathbb{R}^3$ and $\nu_i \in \mathbb{R}^3$. The pose inertia tensor of each body is denoted by the diagonal matrix $\mathbb{I}_i$ defined by

$$\mathbb{I}_i = \begin{pmatrix} \mathbb{J}_i & 0 \\ 0 & m_i I_3, \end{pmatrix}$$

where $\mathbb{J}_i$ is the rotational inertia tensor, m_i is its mass, and I_n denotes the n-x-n identity matrix. The system has n joints described by parameters $r \in \mathbb{R}^n$. Following standard notation [158], the relative transformation between the base body#0 and body#i is denoted by $g_{0i} : \mathbb{R}^n \to \mathrm{SE}(3)$, i.e.

$$g_i = g_0 g_{0i}(r).$$

The control inputs $u \in U \subset \mathbb{R}^{m=n+4}$ denote the four rotor speeds squared and the n joint torques. More specifically, $u_i = \Omega_i^2$ for $i = 1, \ldots, 4$ where Ω_i is the rotor speed of the i-th rotor, and u_{4+i} denotes the i-th joint torque, for $i = 1, \ldots, n$.

The *configuration* of the system is thus given by $q \triangleq (g, r) \in Q \triangleq SE(3) \times \mathbb{R}^n$, where $g \in SE(3)$ is a chosen reference frame moving with the robot. In this work we take the base body as a moving reference, i.e. $g \equiv g_0$. The *velocity* of the system is given by $v \triangleq (V, \dot{r}) \in \mathbb{R}^{6+n}$, where $V \in \mathbb{R}^6$ denotes the body-fixed velocity of the moving frame g and $\dot{r} \in \mathbb{R}^n$ denotes the joint angle velocities. The base velocity satisfies $\widehat{V} = g^{-1}\dot{g}$ where the "hat" operator $\widehat{V}$ for a given $V = (\omega, \nu)$ is defined by

$$\widehat{V} = \begin{bmatrix} \widehat{\omega} & \nu \\ 0_{1\times 3} & 0 \end{bmatrix}, \quad \widehat{\omega} = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}. \tag{3.1}$$

With these definitions, the full *state* of the system is $x \triangleq (q, v) \in X \triangleq Q \times \mathbb{R}^{6+n}$.

Continuous Equations of Motion The coordinates for our setting are $q = (g, r)$ where the pose $g \in SE(3)$ and r represents joint parameters. For optimal control purposes, it is necessary to avoid Euler angle singularities and, in addition, it is advantageous to avoid unit quaternion constraints. To achieve this, the dynamics is defined directly on state space X as:

$$\dot{g} = g\widehat{V} \tag{3.2}$$

$$M(r)\dot{v} + b(q, v) = Bu, \tag{3.3}$$

where the mass matrix $M(r)$, *bias* term $b(q, v)$, and constant control matrix B are computed analogously to standard methods such as the articulated composite body algorithm [55] or using spatial operator theory [87]. With our coordinate-free approach the mass matrix in fact only depends on the shape variables r rather than on q and for tree-structured systems can be computed readily according to

$$M(r) = \left[\begin{array}{c|c} \mathbb{I}_0 + \sum_{i=1}^{n} A_i^T \mathbb{I}_i A_i & \sum_{i=1}^{n} A_i^T \mathbb{I}_i J_i \\ \hline \sum_{i=1}^{n} J_i^T \mathbb{I}_i A_i & \sum_{i=1}^{n} J_i^T \mathbb{I}_i J_i \end{array}\right] \tag{3.4}$$

using the adjoint notation $A_i := \mathrm{Ad}_{g_{0i}^{-1}(r)}$, and jacobian given by

$$J_i := \sum_{j=1}^{n} [g_{0i}^{-1}(r) \partial_{r_j} g_{0i}(r)]^{\vee}, \tag{3.5}$$

where $g_{0i}(r)$ is the relative transformation from the base body to body #i and $\mathbb{I}_i$ is the inertia tensor of body #i [158].

The bias term $b(q, v)$ encodes all Coriolis, centripetal, gravity, and external forces. Finally, for a quadrotor model the constant control matrix B has the form

$$B = \left[\begin{array}{cccc|c} 0 & -lk_t & 0 & lk_t & \\ -lk_t & 0 & lk_t & 0 & \\ k_m & -k_m & k_m & -k_m & \\ 0 & 0 & 0 & 0 & 0_{6\times\ell} \\ 0 & 0 & 0 & 0 & \\ k_t & k_t & k_t & k_t & \\ \hline \multicolumn{4}{c}{0_{\ell\times 6}} & I_{\ell\times\ell} \end{array}\right],$$

where l, k_t, k_m correspond to the distance of the propellers from the center of quadrotor, thrust gain and moment gain respectively and are known constants. This can be easily extended to other multi-rotor configurations.

Discrete Dynamics For computational purposes we employ discrete-time state trajectories $x_{0:N} \triangleq \{x_0, \ldots, x_N\}$ at equally spaced times $t_0, \ldots, t_N \equiv t_f$ with time step $\Delta t = \frac{t_f - t_0}{N}$. The discrete state at index k approximates the continuous state at time $t_k = t_0 + k\Delta t$, i.e. $x_k \approx x(t_k)$ and is defined by $x_k = (g_k, r_k, V_k, \Delta r_k)$, where Δr_k denotes the joint velocities at k-th stage. A simple discrete-time version of the continuous dynamics (3.2)–(3.3) is then employed:

$$g_{k+1} = g_k \,\mathrm{cay}\left(\Delta t V_{k+1}\right), \tag{3.6}$$

$$r_{k+1} = r_k + \Delta t \Delta r_{k+1}, \tag{3.7}$$

$$M(r_k)\frac{v_{k+1} - v_k}{\Delta t} + b(q_k, v_k) = Bu_k, \tag{3.8}$$

where $b(q_k, v_k)$ is the bias term evaluated at kth step. This is a first-order semi-implicit method since one first updates the velocity v_{k+1} using the dynamics (3.8) and then updates the configuration using the kinematics (3.6)–(3.7). The method requires small time-steps to ensure stability ($\Delta t \leq 100$ms is sufficient for the aerial systems considered), higher-order methods are also possible [106, 104].

Note that the base pose update (3.6) is performed using the Cayley map $\text{cay} : \mathbb{R}^6 \to SE(3)$ defined (see e.g. Kobilarov and Marsden [106]) by

$$\text{cay}(V) = \begin{bmatrix} I_3 + \frac{4}{4+\|\omega\|^2}\left(\widehat{\omega} + \frac{\widehat{\omega}^2}{2}\right) & \frac{2}{4+\|\omega\|^2}(2I_3 + \widehat{\omega})\,\nu \\ 0 & 1 \end{bmatrix}, \tag{3.9}$$

instead of the more standard exponential map on $SE(3)$ [158, 27] since it is an accurate and efficient approximation, i.e. $\text{cay}(V) = \exp(V) + O(\|V\|^3)$, it preserves the group structure, and has particularly simple to compute derivatives. Its inverse is denoted by $\text{cay}^{-1} : SE(3) \to \mathbb{R}^6$ and is defined for a given $g = (R, p)$, with $R \neq -I$, by

$$\text{cay}^{-1}(g) = \begin{bmatrix} [-2(I+R)^{-1}(I-R)]^{\vee} \\ (I+R)^{-1}p \end{bmatrix}.$$

3.3 Trajectory Optimization

To achieve agile pick-and-place motions we employ model-predictive control to optimize future trajectories over the interval $[t_0, t_f]$ where t_0 is the current time and t_f is a specified moving horizon. A typical horizon $t_f - t_0$ for the considered aerial maneuvers is between 2 and 5 seconds. Two methods for unconstrained optimal control are considered in view of their capacity for near real-time performance: a simple Gauss-Newton shooting method and a Stagewise Newton sweep method.

3.3.1 Optimal Control Formulation

The optimal control problem can be generally formulated as the minimization of:

$$J(x_{0:N}, u_{0:N-1}) \triangleq L_N(x_N) + \sum_{k=0}^{N-1} L_k(x_k, u_k), \tag{3.10}$$

$$\text{subject to:} \qquad x_{k+1} = f_k(x_k, u_i), \qquad u_k \in U \tag{3.11}$$

where f_k encodes the integrator (3.6)–(3.8) and U defines the admissible control set. The stage-wise cost penalizes deviation from a desired nominal state x_d and controls u_d and is given by

$$L_k(x_k, u_k) = \frac{1}{2}\|x_k - x_d\|^2_{Q_k} + \frac{1}{2}\|u_k - u_d\|^2_{R_k}, \tag{3.12}$$

while the terminal cost is defined by

$$L_N(x_N) = \frac{1}{2}\|x_N - x_f\|^2_{Q_f}, \tag{3.13}$$

where $Q_k \geq 0, Q_f > 0, R_k > 0$ are appropriately chosen diagonal matrices to tune the vehicle behavior while reaching a desired final state x_f. In the aerial robot application the matrix Q_k contains non-zero terms corresponding to a desired velocity only.

3.3.2 Overloading $\pm$operator on the group SE(3)

Numerical optimal control is based on vector calculus which is not directly applicable to states $x = (g, r, v)$ containing matrix elements $g \in SE(3)$. Hence, we use vector operators with analogous "retract" and "lift" operators on $SE(3)$.

The *lift* operator on $SE(3)$ is equivalent to operator *minus* $(\cdot) - (\cdot) : SE(3) \times SE(3) \to \mathbb{R}^6$

$$g_b - g_a = V \quad \Longleftrightarrow \quad \mathrm{cay}^{-1}(g_a^{-1} g_b) = V,$$

$$g_b - g_a \triangleq \mathrm{cay}^{-1}(g_a^{-1} g_b) = V \in \mathbb{R}^6, \tag{3.14}$$

or practically speaking the differences between two poses approximately equals the constant body-fixed velocity V with which g_a moves to align with g_b after one unit of time. The *retract* operator on $SE(3)$ is equivalent to *plus* or *minus* $(\cdot) \pm (\cdot) : SE(3) \times \mathbb{R}^6 \to SE(3)$ according to

$$g_a \pm V \triangleq g_a \mathrm{cay}(\pm V) = g_b \in SE(3), \tag{3.15}$$

i.e. adding/subtracting a vector V to/from the matrix g_a is interpreted as shifting g_a using a unit-time transformation with constant body-fixed velocity V. With these definitions, the errors $x_i - x_d$ and $x_N - x_f$ appearing in the costs (3.12),(3.13) are defined using the lift operator (3.14) so that, e.g. the latter with $x_f = (g_f, r_f, v_f)$ should be understood as

$$x_N - x_f \equiv \begin{bmatrix} \mathrm{cay}^{-1}(g_f^{-1} g_N) \\ r_N - r_f \\ v_N - v_f \end{bmatrix}.$$

3.3.3 Gauss-Newton shooting method

One of the simplest, often overlooked, but surprisingly effective methods for solving the optimal control problem (3.10)–(3.11) is a shooting method exploiting the least-squares nature of the costs (3.12)–(3.13). It is formulated by parametrizing the discrete control trajectory $u_{0:N-1}$ using a vector $\xi \in \mathbb{R}^{\ell \le Nm}$, encoded through the functions $u_k = \phi_k(\xi)$ for each $k = 0, \dots, N-1$. For instance, ξ could contain the knots of a B-spline from which each u_k is extracted. The simplest parametrization is to simply set $\xi = u_{0:N-1}$. Using the dynamics each state can be expressed as a function of ξ which is encoded through the functions $x_k = \psi_k(\xi)$ for $k = 0, \dots, N$. The cost is then expressed as $J(\xi) = \frac{1}{2}h(\xi)^T h(\xi)$, where $h : \mathbb{R}^\ell \to \mathbb{R}^{N(m+n+6)}$ is given by

$$h(\xi) = \begin{bmatrix} \sqrt{R_0}\,(\phi_0(\xi) - u_d) \\ \sqrt{Q_1}\,(\psi_1(\xi) - x_d) \\ \sqrt{R_1}\,(\phi_1(\xi) - u_d) \\ \vdots \\ \sqrt{Q_{N-1}}\,(\psi_{N-1}(\xi) - x_d) \\ \sqrt{R_{N-1}}\,(\phi_{N-1}(\xi) - u_d) \\ \sqrt{Q_f}\,(\psi_N(\xi) - x_f) \end{bmatrix}.$$

Since $R_i > 0$ the Jacobian $\partial h(\xi)$ is guaranteed to be full rank and one can apply a Gauss-Newton iterative method directly to update $\xi \to \xi + \delta\xi$ where $\delta\xi = -(\partial g^T \partial g)^{-1} \partial g^T g$. In addition, the Jacobian has a lower-triangular structure that can be exploited in the Cholesky GN solution. The complexity of this method is still $O(\ell^3)$ which is only acceptable for small ℓ, e.g. $\ell \le 100$ in order to achieve real-time performance. The key advantage of the GN approach is its simplicity and robustness by employing standard regularization and line-search techniques [19].

A more efficient method with complexity $O(N(n+m))$ that exploits the recursive optimal control problem structure is presented next.

3.3.4 Stagewise Newton and Differential Dynamic Programming

The second optimal control method used in this work is based on a coordinate-free recursive optimal control formulation [104, 106] for optimization on state spaces with Lie group structure such as SE(3). The particular method we employ is Stagewise Newton (SN) [19] which is also closely related to Differential dynamic programming (DDP) [85].

Stagewise methods explicitly require the linearization of the cost and of the dynamics. On non-Euclidean manifolds X such linearization is achieved using *trivialized* variations and gradients [104]. In particular, for the class of systems considered in this work, the linearized discrete dynamics takes the form

$$dx_{k+1} = A_k dx_k + B_k \delta u_k, \tag{3.16}$$

with $dx_k \triangleq (dg_k, \delta r_k, \delta v_k)$ where $dg \triangleq (g^{-1}\delta g)^\vee \equiv ((R^T \delta R)^\vee, R^T \delta p) \in \mathbb{R}^6$ is the *trivialized variation* on SE(3). Similarly, the trivialized gradient $d_g L \in \mathbb{R}^6$ of a function $L : SE(3) \to \mathbb{R}$ is defined by

$$d_g L \triangleq \nabla_V \Big|_{V=0} L(g\,\mathrm{cay}(V)), \tag{3.17}$$

for some $V \in \mathbb{R}^6$. With these definitions, any standard iterative optimization method such as SQP, SN, or DDP can be applied by replacing the standard gradients $\nabla_g L$, $\nabla_g^2 L$ and variations δg, with the trivialized gradients $d_g L$, $d_g^2 L$ and trivialized variations dg.

Finite-difference linearization of the dynamics. Since the resulting multi-body dynamics (3.6)–(3.8) has a complex nonlinear form, we employ finite differences for computing the Jacobians A_k and B_k. The default choice is central differences:

$$A_k^i \approx \frac{f(x_k + \epsilon e_i, u_k) - f(x_k - \epsilon e_i, u_k)}{2\epsilon},$$

$$B_k^j \approx \frac{f(x_k, u_k + \epsilon e_j) - f(x_k, u_k - \epsilon e_j)}{2\epsilon},$$

for $i = 1, \ldots, n+6$, and $j = 1, \ldots, m$, where each e_i is a standard basis unit vector with only one non-zero element at its i-th component. We again emphasize that the $+$ and $-$ signs above should be interpreted as the overloaded operators (3.14),(3.15) whenever elements of SE(3) are involved.

Closed-form cost gradients. The trivialized gradient and Hessian of L_i are straightforward to compute and only require an extra term to account for the Cayley map. They are given by:

$$dL_k = \begin{bmatrix} \mathrm{dcay}^{-1}(-\Delta_k) & 0 \\ 0 & I \end{bmatrix}^T Q_k (x_k - x_d), \tag{3.18}$$

$$d^2 L_k \approx \begin{bmatrix} \mathrm{dcay}^{-1}(-\Delta_k) & 0 \\ 0 & I \end{bmatrix}^T Q_k \begin{bmatrix} \mathrm{dcay}^{-1}(-\Delta_k) & 0 \\ 0 & I \end{bmatrix}, \tag{3.19}$$

where $\Delta_k = \text{cay}^{-1}(g_d^{-1} g_k)$ for each $k = 0, \dots, N-1$. Equivalent expressions also hold for the gradients of L_N, with g_d replaced by g_f, and Q_k with Q_f. Note that for simplicity the Hessian was approximated by ignoring the second derivative of cay. The trivialized Cayley derivative denoted by dcay(V) for some $V = (\omega, v) \in \mathbb{R}^6$ is defined (see e.g. Kobilarov and Marsden [106]) as

$$\text{dcay}(V) = \begin{bmatrix} \frac{2}{4+\|\omega\|^2}(2I_3 + \widehat{\omega}) & 0_3 \\ \frac{1}{4+\|\omega\|^2}\widehat{v}(2I_3 + \widehat{\omega}) & I_3 + \frac{1}{4+\|\omega\|^2}(2\widehat{\omega} + \widehat{\omega}^2) \end{bmatrix}, \tag{3.20}$$

it is invertible and its inverse has the simple form

$$\text{dcay}^{-1}(V) = \begin{bmatrix} I_3 - \frac{1}{2}\widehat{\omega} + \frac{1}{4}\omega\omega^T & 0_3 \\ -\frac{1}{2}\left(I_3 - \frac{1}{2}\widehat{\omega}\right)\widehat{v} & I_3 - \frac{1}{2}\widehat{\omega} \end{bmatrix}. \tag{3.21}$$

The linearized dynamics (3.16), cost gradients (3.18) and Hessians (3.19) can now be used as the ingredients of a standard Stagewise Newton algorithm [19] as detailed in Kobilarov [104].

3.4 Experiment Setup

In this section the hardware and software architecture required for running the manipulation experiments is described. The goal of the experiments is to pick a bottle of weight 100 grams from a modified desk that is approximately 1 meter away. The aerial manipulator achieves this task by either following a kinematic reference trajectory or an optimal reference trajectory. A qualitative comparision of the time taken to grasp the object and quality of tracked trajectories are compared.

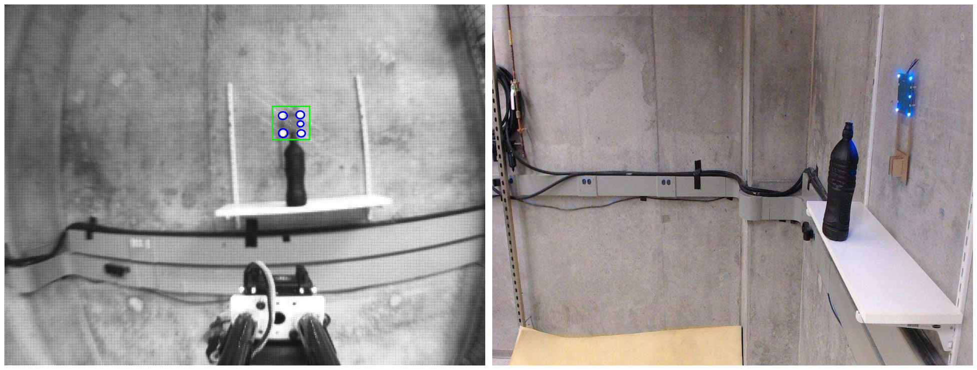


Figure 3.2: a) Experimental arena showing the object to grab (black bottle) and led markers b) Led markers as seen from onboard camera

3.4.1 Hardware

Our prototype platform is based on the 3DRobotics quadcopter capable of lifting a payload of 1Kg, the Pixhawk autopilot board [150] for low-level attitude and thrust control, and the Odroid XU+E bare board computer for running various control algorithms. The NaturalPoint OptiTrack Motion Capture System has been used for estimating the attitude and position of the quadcopter in the world frame. A lightweight camera (PointGrey Firefly model) is installed onboard for providing the relative position of the target object in the reference frame of the quadcopter. A custom manufactured lightweight arm along with a 3D printed gripper has been installed on the quadcopter to grasp the object.

3.4.2 Bridging Gap Between Trajectory Generation and Hardware Inputs

The trajectory generation procedure described in section 3.3 used body torques to control the quadrotor and joint torques to control the robotic arm. In practice, however, we use an autopilot to control the orientation of the quadrotor and servos to control the joint angles. Thus, we use the states provided by the trajectory optimization as a reference for a nonlinear controller. Using an optimal reference trajectory is expected to improve the tracking performance of the nonlinear controller as opposed to a kinematic reference trajectory. Further, by delegating the low-level control to the autopilot, we can switch between different configurations of multirotor platforms without modifying the reference trajectory.

3.4.3 Experimental Scenario

The experimental scenario in Figure 3.2 shows the object of interest and LED markers which are used by the onboard camera to detect and track the object. The experiment requires the quadcopter to detect the marker, fly to a specified location in front of it, and retrieve the object placed on a stand. This is a challenging task, since the quadcopter has to extend the arm farther beyond it's enverlope to grab the target object.

3.4.4 Kinematic reference trajectory generation

According to kinematic trajectory generation approach, we have two stages in the process of grabbing the object. In the first stage, we track the markers

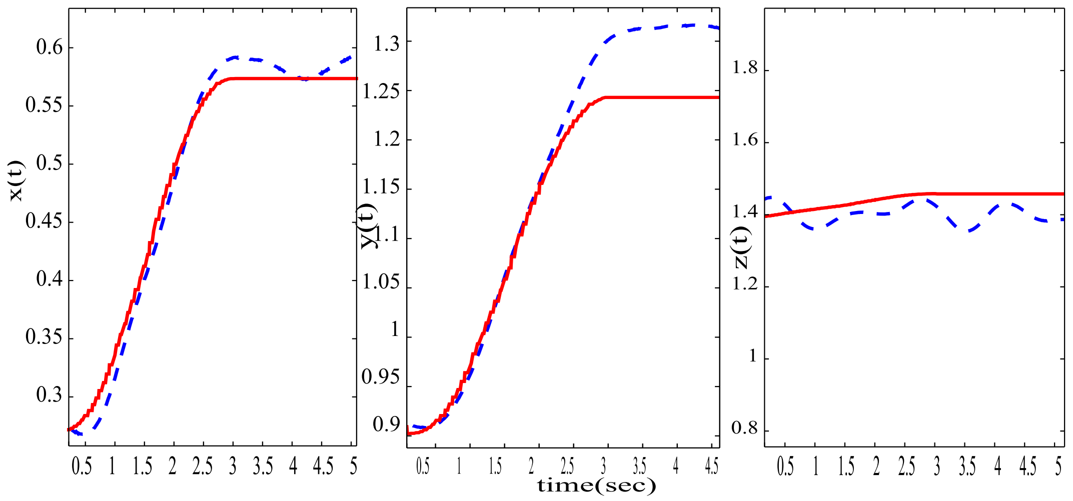


Figure 3.3: Optimal control trajectory followed by the quadcopter. The solid line represents the desired trajectory and dashed line represents the actual trajectory followed

and stabilize to 0.6m away from it. Once the quadcopter stabilizes to the goal position, we begin the next stage in which the arm is opened at a constant velocity until the object is grasped.

3.4.5 Optimal reference trajectory generation

Using trajectory optimization approach, we compute a reference trajectory for the combined system of the quadcopter and arm using Stagewise Newton method described in section 3.3.4. The optimal controller is used in open-loop to compute a reference trajectory for the quadcopter and the arm. The object position is assumed to be static once the quadcopter starts executing the reference trajectory. The computed trajectory contains the full state (position, orientation and body fixed velocities) of the quadcopter, full state of the arm (joint positions and velocities) and the controls needed to achieve them.

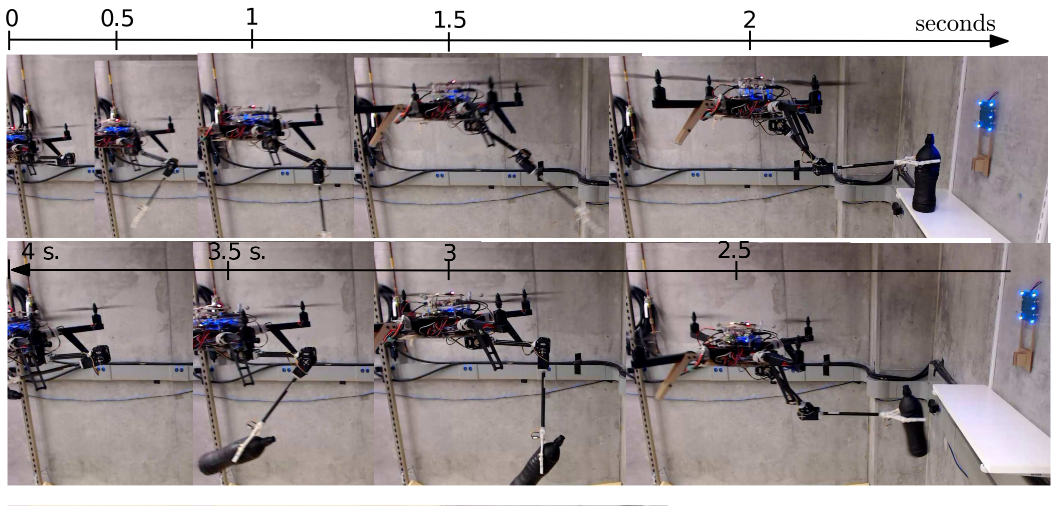


Figure 3.4: Series of pictures showing approach, grasping, and retrieval of the object under optimal reference trajectory setting

The desired position and velocity of the quadcopter is fed into the feedback linearization based controller and the desired joint angles and velocities for the arm are achieved through the PID controller on the servo motors. The quadcopter is able to track the trajectory closely as shown in Figure 3.3. The optimal control approach allows for faster actuation of the arm without losing accuracy. Since both the arm and the quadcopter execute their respective trajectories simultaneously, the object is expected to be retrieved in a shorter time interval as compared to kinematic reference generation.

3.5 Results and Discussion

The series of pictures in Figure 3.4 show the quadcopter flying to the marker and retrieving the object. The upper half represents the approach to grasp the object and the lower half shows the retrieval of the object. The average time for retrieval using kinematic reference trajectories (Fig[3.5]) is around 15 seconds. Using optimal reference trajectory tracking reduces the time taken to grasp the object to 5 seconds. This suggests that using optimal reference trajectories is superior to manual reference trajectories.

The tip positions for various starting positions converging to the target location (red cuboid) have been plotted in Figure 3.5. The tip positions are defined as the position of the end-effector of the manipulator attached to the quadrotor. The tip position is obtained by combining motion capture pose of the quadrotor and the forward kinematics of the manipulator. Since the non-optimal kinematic reference trajectory does not account for the dynamics of the arm, the tip positions are not smooth and take longer time to converge to the grasping location. On the other hand, the tip position for the case using optimal control (denoted by black dashed line) is smoother and converges with the same accuracy in a shorter time period. Following the optimal control based reference trajectory has enabled us to actuate both the quadcopter and the arm simultaneously to grasp the object quickly.

There are many challenges faced during the manipulation tasks described above. The quadcopter position can be estimated based on the pose of the markers from the onboard camera. But this estimate turned out to be noisy and is dependent on the distance between the camera and markers. Thus a motion capture system is used to provide a reliable estimate of the quadcopter state. Since we are using a feedback linearization based controller for the quadcopter, we are not using optimal control to its full capacity of directly commanding the quadcopter motors. This explains the slight discrepancies between the actual and the desired quadcopter trajectories shown in Figure 3.3.

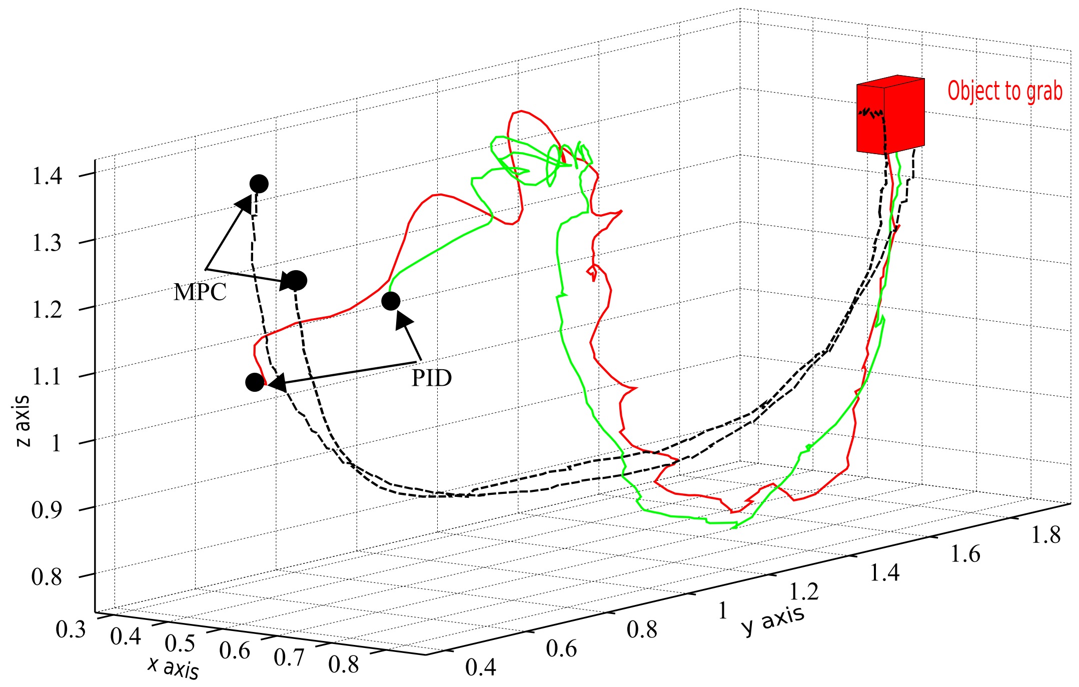


Figure 3.5: Comparison of end-effector tip position trajectories for Kinematic and MPC settings starting from from different initial positions marked with black dots.

3.5.1 Future Directions

Future work should focus on two aspects: model identification, motor-level control. Model identification should be performed online to identify the mass of the object that's being picked up and learn the inertial parameters of the system. From the author's experience getting noise free IMU measurements at high frequency is critical for learning inertial parameters of the system. It was observed that the IMU signal was drowned out by vibration from propellers to figure out the moment of inertia of the system. Motor-level control will allow the MPC controller to track reference trajectories more closely by countering the torques applied by the arm and the picked up object. Without motor-level control it is not possible to lift very heavy objects since the autopilot on the quadrotor is not designed to handle abrupt change in external torques.

Chapter 4

Safe Obstacle Avoidance

In this Chapter, we develop techniques to handle obstacles and other task constraints safely under the influence of external disturbances and inaccurate dynamic models. We introduce two techniques to perform obstacle avoidance. The first technique introduced is called Adaptive NMPC which propagates parametric uncertainty to find approximate high confidence ellipsoids along the state trajectory. The trajectory optimization then avoids obstacles using the high confidence ellipsoids as a buffer to ensure the robot does not collide with obstacles. This method only propagates the uncertainty in open-loop and a second method called Tube NMPC is introduced which takes into account re-planning through a feedback controller. We also provide simulations and experiments to prove the usefulness of these methods.

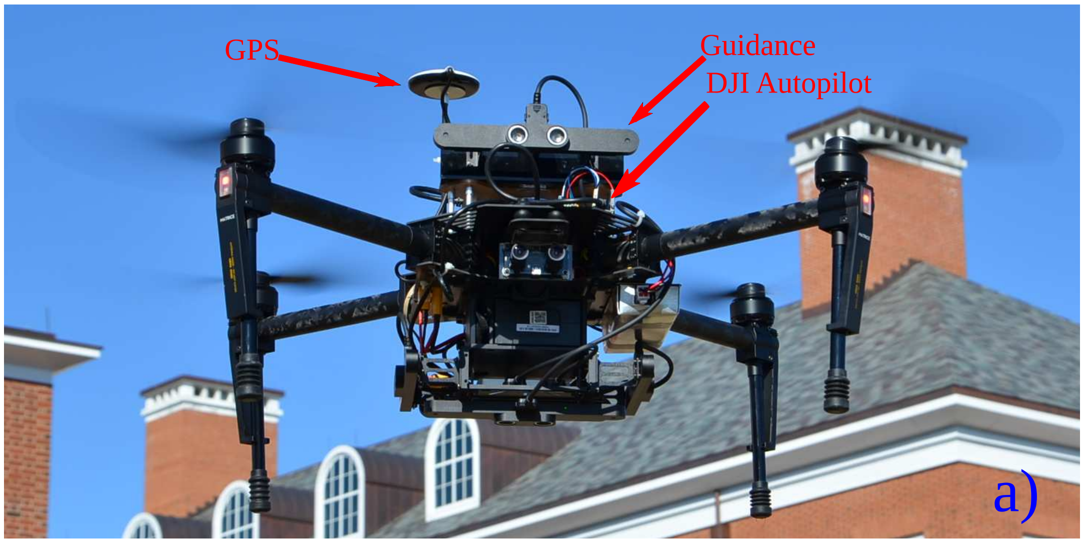
4.1 Adaptive Nonlinear Model Predictive Control

4.1.1 Introduction

This section introduces a robust control scheme for multirotors to perform obstacle avoidance. Robustness has been achieved by using an online estimation scheme to learn the dynamics of a quadrotor and steering away from obstacles with high probability by predictng the vehicle motion for a short time horizon into the future.

Aerial robotic vehicles such as quadrotors are beginning to enable a range of useful capabilities. Current and future applications of quadrotors operating in natural environments include delivery of packages [184], inspection of a building infrastructure and power lines [3], aerial photography, and traffic surveillance [68]. With the increasing use of quadrotors, their safety and reliability are becoming essential.

Several prior studies focused on generating reliable controllers for guaranteeing safety and stability of the quadrotor under external disturbances. For instance, the quadrotor dynamics has been estimated online using the autoregressive-moving-average with exogenous inputs (ARMAX) model [202, 199] and has been applied to model and control quadrotor platforms [201]. Model Reference Adaptive Control (MRAC) algorithms that estimate the system parameters while stabilizing the quadrotor under actuator uncertainty have been developed in [51, 161, 93]. These methods, however, do not account for modeling uncertainty and actuator bounds. Robust control techniques, on the other hand, can reject uncertainty in system dynamics [231] and account



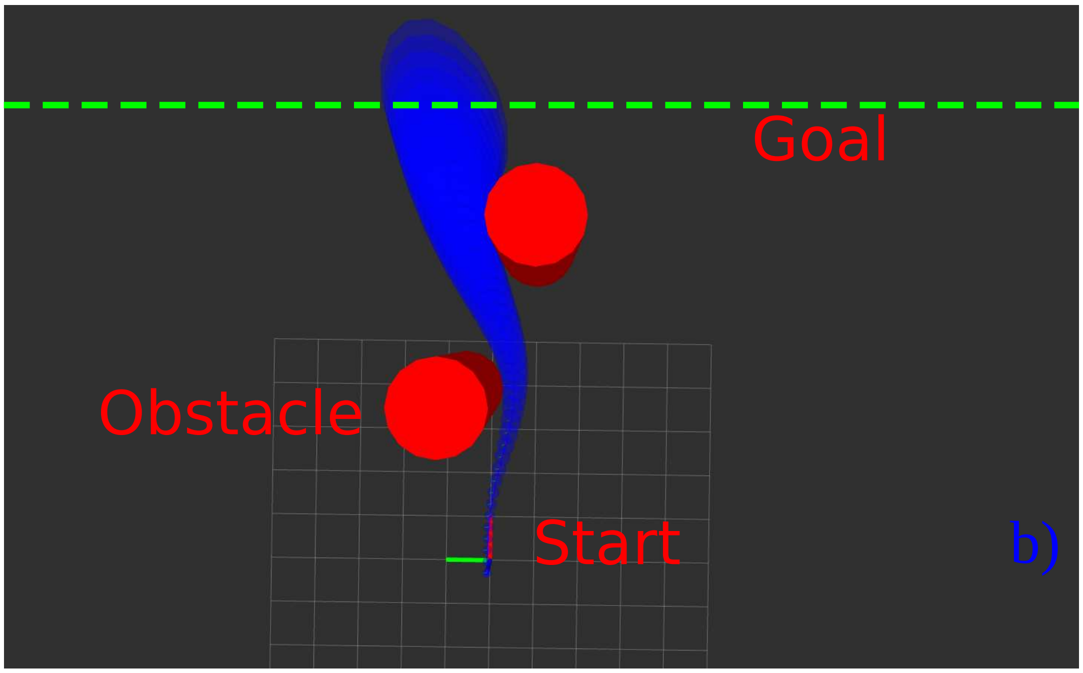


Figure 4.1: a) DJI Matrice quadrotor with Guidance sensor suite and an additional short-range stereo used for experiments, b) Simulated safe obstacle avoidance trajectory for the quadrotor. The blue region corresponds to propagated uncertainty denoted by a 2σ standard deviation ellipsoids around the pose of the quadrotor, and red regions show two cylindrical obstacles.

for actuator limits [42]. A robust backstepping controller for quadrotors which is globally asymptotically stable has been discussed in Kobilarov [107]. Robust controllers have been used in practice to track an aggressive figure eight maneuver accurately [134].

Introducing obstacles into the quadrotor's environment further complicates the safe operation of quadrotors. Potential field methods have been used in Budiyanto et al. [25] to generate an optimal obstacle free path that handles both static and dynamic obstacles. This method uses only the kinematics of the quadrotor model to generate the trajectory. On the contrary, a Linear Quadratic Regulator (LQR) based feedback scheme, combined with a direct collocation-based obstacle avoidance planner has been used to plan dynamically feasible knife-edge maneuvers for fixed wing aircraft in Barry et al. [16]. The feedback system generates a time-varying locally stable feedback control law using LQR optimization. Instead of offline generated feedback controllers, real-time kinodynamic planning using sampling-based motion planning of a quadrotor among dynamic obstacles has been demonstrated in Allen and Pavone [6].

Obstacle avoidance in the presence of external disturbances is dealt with using recent extensions in Learning-Based Model Predictive Control (LBMPC) [24, 15]. This method allows for online learning of quadrotor dynamics and the generation of optimal trajectories that guarantee convergence. LBMPC has been used to learn ground effects on quadrotors and to predict the trajectory of balls for the purpose of catching them. LBMPC has also been used in controlling other systems with unknown dynamics such as a 3DOF robot arm [126] and the energy management of air conditioners in a building [14]. This method linearizes the dynamics and solves the MPC problem formulated as a quadratic program in an efficient onboard implementation.

Robust motion planning for fixed wing aircraft and quadrotors under the influence of parametric uncertainty and external disturbances has been studied recently by Majumdar and Tedrake [141]. This work computes an offline library of feedback funnels to provide a reliable system that will remain inside the funnel when the feedback law is executed. The feedback funnels are sequenced together in real time to avoid obstacles reactively based on their positions. This method has the drawback that it does not account for online changes in model parameters and deviations in the uncertainty of external disturbances.

This section focuses on robust obstacle avoidance of multi-rotor platform in an outdoor scenario with varying external disturbances. A nonlinear stochastic quadrotor model incorporating external disturbances is learned online. The learned model has been employed in a novel Nonlinear Model Predictive Control (NMPC) optimization framework that plans quadrotor trajectories to avoid obstacles by a required safety margin. The safety margin is computed based on propagating the model parameter uncertainty and initial state uncertainty as high-confidence ellipsoids in pose space and minimizing the ellipsoid penetration into the obstacles while minimizing control effort, and achieving a user-specified goal location. Figure 4.1 shows an optimal trajectory for a quadrotor avoiding obstacles. This method propagates the uncertainty due to initial state measurements, model parameters, and external disturbances to uncertainty in state trajectory. Sensor uncertainty is not accounted for in this method.

The proposed method is applied towards reactive avoidance of virtual obstacles in an outdoor scenario (a large field with external wind and no motion capture system). Virtual obstacles are referred to as obstacles on a virtual map that the quadrotor is assumed to be flying in. The obstacles are detected online using a virtually-rendered image and a safe trajectory is planned and executed reactively. Additionally, the ability of the learned model to predict the quadrotor's state is verified through multiple experimental trials.

The rest of the section is organized as follows. In section 4.1.2, a simplified nonlinear model of quadrotor dynamics that is appropriate for system identification is proposed. Further, the parameters for the nonlinear model are identified using an online setup of a maximum likelihood estimation framework. In section 4.1.3, an NMPC based optimization scheme is discussed that takes into account uncertainty in state space explicitly and plans trajectories that avoid obstacles by a safety margin. Finally, in section 4.1.4, the results of avoiding an obstacle using the current framework are demonstrated.

4.1.2 System Identification

The quadrotor is modeled as a rigid body attached with four axially aligned rotors. The rotors apply a thrust force along a known body fixed axial direction and torques along three mutually perpendicular body axes. The quadrotor is usually equipped with an autopilot module that converts commanded Euler angles into rotor velocities using a linear Proportional-Integral-Derivative

(PID) controller [65]. The goal of this section is to propose a second-order closed-loop model that models both the rigid body dynamics and the autopilot control loop.

The state of the quadrotor system is given by the position p, rotation matrix R, velocity $\dot{p}$ measured with respect to an inertial frame and angular velocity ω in body frame. The rotation matrix is decomposed into body Euler angles as $R(\xi) = e^{\xi_3 \widehat{e}_3} e^{\xi_2 \widehat{e}_2} e^{\xi_1 \widehat{e}_1}$. The inputs for the second order closed-loop model are the commanded rate of body Euler angles as $\dot{\xi}_c$ and the commanded thrust u_t. A quadrotor model emulating a second order rotational dynamics has been proposed in Eq (4.1). Similar simplified models have been used in system identification of quadrotors [201, 24].

$$\frac{d}{dt}\begin{bmatrix} p \\ R \\ \dot{p} \\ \omega \\ \xi_c \end{bmatrix} = \underbrace{\begin{bmatrix} \dot{p} \\ R\widehat{\omega} \\ g + a_e \\ -k_p(\xi - \xi_c) - k_d\dot{\xi} + \alpha_e \\ 0 \end{bmatrix}}_{\mathbf{f}(\mathbf{x},\text{`})} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ k_t R e_3 & 0 \\ 0 & k_d \\ 0 & 1 \end{bmatrix}}_{\mathbf{g}(\mathbf{x},\text{`})} \begin{bmatrix} u_t \\ \dot{\xi}_c \end{bmatrix}. \tag{4.1}$$

The hat operator $\widehat{\cdot}$ maps a vector in $\mathbb{R}^3$ to $se(3)$ as shown in Eq (3.1). The unknown parameters for the model are the proportional and derivative gains k_p, k_d, thrust gain k_t, and external acceleration a_e and torques α_e ($\theta = [k_p^T, k_d^T, k_t, a_e^T, \alpha_e^T]^T$). Using position and orientation measurements of the quadrotor and assuming the parameters and the measurements are distributed according to a Gaussian distribution, standard MLE techniques can be applied to find the unknown parameters under sufficient excitation [44]. For the quadrotor system, exciting the quadrotor in roll, pitch, and yaw directions with constant thrust is sufficient to estimate the parameters. Figure 4.2

compares the predicted Euler angles and body angular velocities with measurements from an Inertial Measurement Unit (IMU) during an experimental flight path.

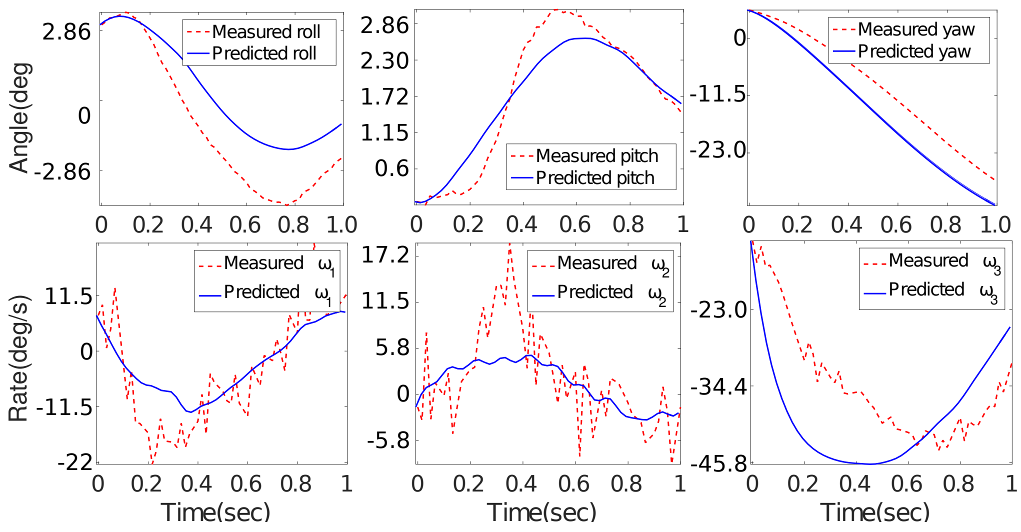


Figure 4.2: The graphs show predicted and observed body Euler angles and body angular rates for the quadrotor on a test data set. The predicted Euler angles are close to the measured values using the second order closed-loop model.

4.1.3 Obstacle Avoidance using Nonlinear Model Predictive Control

In this section, an optimization scheme is designed to produce NMPC trajectories that avoid obstacles using the quadrotor model identified in (4.1). Uncertainty in the model is taken into account by planning a trajectory that stays away from obstacles by a safety margin based on the uncertainty propagation from the estimated parameters to trajectory uncertainty.

4.1.3.1 Discrete Dynamics

NMPC optimization requires a discrete version of the dynamics specified in Eq (4.1). The control inputs to the second order closed-loop model (i.e. the commanded thrust u_t and commanded Euler angle rates $\dot{\xi}_c$) are assumed to be constant during a time step of duration h. The discrete state consisting of position p_i, velocity v_i, rotation matrix R_i, and body angular velocity ω_i is propagated as

$$\underbrace{\begin{bmatrix} p_{i+1} \\ R_{i+1} \\ v_{i+1} \\ \omega_{i+1} \\ \xi_{c_{i+1}} \end{bmatrix}}_{x_{i+1}} = \underbrace{\begin{bmatrix} p_i + \frac{1}{2}(v_i + v_{i+1})h \\ R_i \exp(\frac{1}{2}(\omega_i + \omega_{i+1})h) \\ v_i + h(g + a_0) \\ \omega_i + h(-k_p(\xi_i - \xi_{c_{i+1}}) - k_d\dot{\xi}_i + \alpha_e) \\ \xi_{c_i} \end{bmatrix}}_{f_i} +$$

$$\underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ k_t R_i e_3 & 0 \\ 0 & k_d \end{bmatrix}}_{g_i} \underbrace{\begin{bmatrix} u_{t_i} \\ \dot{\xi}_{c_i} \end{bmatrix}}_{u_i}$$

The linear and angular velocities v_{i+1}, ω_{i+1} are propagated first. The average linear and angular velocities in turn are used in the position and orientation updates using a semi-implicit scheme in the above equation.

4.1.3.2 Propagating Uncertainty

The uncertainty of parameters obtained from MLE is propagated to the uncertainty in the states using the unscented transform as explained in Chapter 2.

The uncertainty in states is then used to plan safe trajectories for avoiding obstacles. The unmodeled dynamics are included in the uncertainty through external accelerations and torques a_e, τ_e. The uncertainty in detecting the obstacles is not included in this propagation scheme.

The uncertainty propagation scheme finds the mean and covariance of the states along the trajectory given the estimated mean and covariance of the unknown parameters in dynamics. The mean and covariance of the estimated parameters θ obtained using MLE estimation are denoted by

$$\theta^* = [k_p^*, k_d^*, k_t^*, \bar{a}_e, \bar{\alpha}_e], \qquad \Sigma_{\theta^*} = [\Sigma_{k_p^*, k_d^*, k_t^*}, \Sigma_{\bar{a}_e}, \Sigma_{\bar{\alpha}_e}]. \tag{4.2}$$

The mean and covariance of the nominal states obtained through unscented transform can be written as

$$(\bar{x}_{0:N-1}, \Sigma_{x_{0:N-1}}) = \mathit{UnscentedTransform}(u_{0:N-1}, \theta^*, \Sigma_{\theta^*})$$

where the $\bar{x}_{0:N-1}$ is the mean state along the trajectory, $\Sigma_{x_{0:N-1}}$ is the covariance of the the state along the trajectory, and $u_{0:N-1}$ is the control trajectory.

Figure 4.3 shows the plot of a predicted trajectory distribution obtained using unscented transforms based on parameters estimated from real data. It can be observed that the navigated trajectory in red falls within the predicted 2σ region in blue, indicating that the model is a good fit for the dynamics. The uncertainty in the predicted trajectory is used to avoid obstacles using an NMPC formulation explained next.

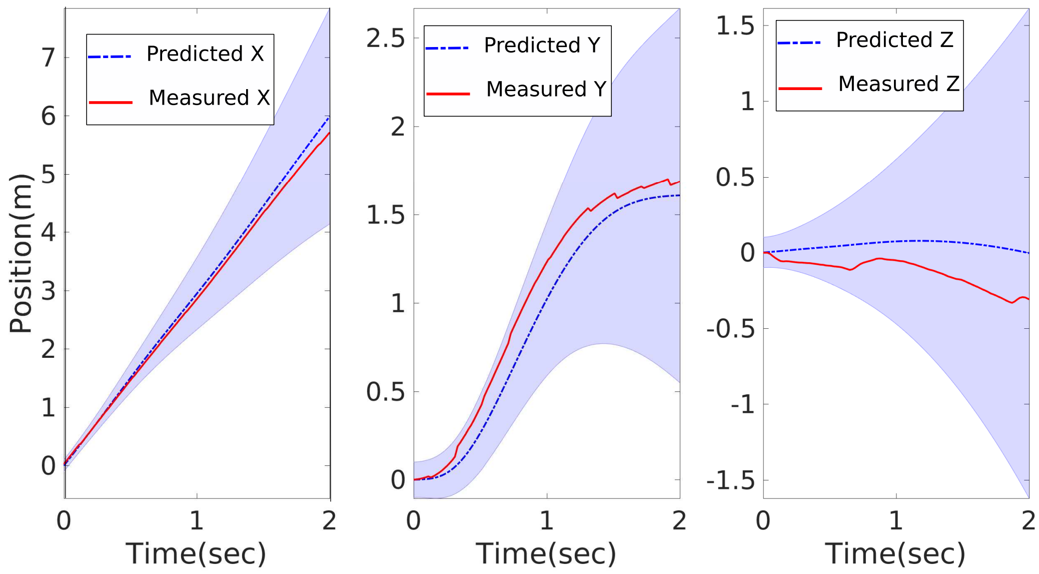


Figure 4.3: The predicted mean position and 2σ region of the quadrotor obtained using unscented transforms is shown in blue. The measured position of the quadrotor is shown in red.

4.1.3.3 NMPC Formulation - Obstacle Avoidance

The NMPC formulation in Chapter 2 is augmented with an additional constraint to ensure obstacle avoidance. We define an inequality constraint for every obstacle at every point along the trajectory to ensure the nominal trajectory avoids the obstacle by buffer dependent on the covariance at stage i. The standard deviation ellipsoid and the obstacle inequality are defined as

$$P_i = \{p : (p - p_i)^T \Sigma_{p_i}^{-1} (p - p_i) \leq k_\sigma^2\}, \tag{4.3}$$

$$d_{i,j} = \text{Dist}(P_i, o_j) \geq 0, \tag{4.4}$$

where the ellipsoid P_i consists of all the points which are within k_σ standard deviations from the current position p_i and the signed distance function *Dist* is designed such that it is negative when the ellipsoid is intersecting with the boundary of the obstacle and is positive otherwise as shown in Figure 4.4.

Distance between Nominal trajectory and Obstacle

The obstacles in this work are assumed to be cylinders with center o_{p_j}, radius o_{r_j} and axis o_{a_j}. To design the distance function, we first project the standard deviation ellipsoid on to the plane perpendicular to the cylindrical axis as explained in Pope [177]. Let us assume the projected ellipsoid is given by

$$\overline{P}_i = \{y : (y - y_i)^T \overline{\Sigma}_{y_i}^{-1} (y - y_i) \leq k_\sigma^2\}, \tag{4.5}$$

where y_i is the vector p_i projected on to the cylindrical plane and $\overline{\Sigma}_{y_i}$ is the projected covariance matrix. The distance function is then found by expanding the projected ellipsoid by the obstacle cylinder radius o_{r_j} and ensuring that the projected center of the obstacle $\overline{o}_{p_j}$ is outside the expanded ellipsoid:

$$Dist(P_i, o_j) = k_\sigma^2 - (\overline{o}_{p_j} - y_i)^T \left(\overline{\Sigma}_{y_i} + o_{r_j}\mathbb{I}\right)^{-1} (\overline{o}_{p_j} - y_i) \geq 0 \tag{4.6}$$

The distance function ensures that all the points which are within k_σ standard deviations from the nominal trajectory do not intersect with the cylinder. Thus by solving the constrainted NMPC optimization, the quadrotor can avoid obstacles with high probability. The size of the ellipsoid is regulated by the inflation paramater k_σ which denotes the confidence interval around the nominal trajectory. In this work, we used k_σ as 2 to make sure the probability of collision is approximately 5%.

The Levenberg–Marquardt algorithm is used to find the optimal controls

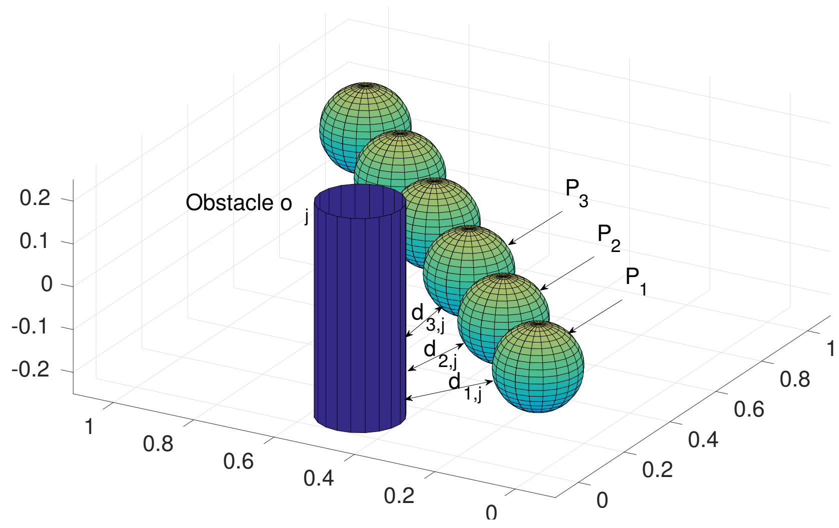


Figure 4.4: Plot of a sample trajectory with standard deviation ellipsoids P_i and the distance to obstacle d_i. The ellipsoids P_i consist of points which are closer than k_σ standard deviations from the quadrotor position.

u^* since the cost functions L_i, L_N 2.5 are chosen to be of least-squares form. To reduce the dimension of the optimization, the control inputs u_i are produced using a uniform B-spline of second order with knots given by ψ.

The obstacle avoidance constraint is enforced as a soft penalty in the residual function. The residual $\sqrt{k_o}D_{i,j}(\psi)$ minimizes the intersection between the standard deviation ellipsoids P_i and the obstacle o_j. The optimization algorithm is iterated multiple times by increasing the obstacle gain k_o after every run. This procedure smoothly transitions from a trajectory with P_i intersecting the obstacle to a trajectory with P_i slightly grazing the obstacle as the gain k_o becomes very large. The complete optimization procedure is listed in algorithm 4

Figure 4.1 shows an example obstacle avoidance scenario, where the quadrotor avoids two cylinders in front of it while flying at 5m/s. The goal

Algorithm 4 Obstacle avoidance using Levernberg-Marquardt

```
Given $\theta^*, \Sigma_{\theta^*}, k_o, \psi_0, \epsilon, k_{o_{max}}, \alpha_{k_o}$
$iters \leftarrow 0$
**for** $iters$ <max_iters **do**
    Compute ellipsoids $\Sigma_{x_{1:N}}$ using *UnscentedTransform*
    Project ellipsoids to position space $\Sigma_{p_{1:N}}$
    Evaluate change in knots $\delta\psi_i$ using *Levenberg-Marquadt* update
    **if** $\|\delta\psi_i\| \leq \epsilon$ **then**
        **if** $k_o > k_{o_{max}}$ **then**
            Exit optimization
        **end if**
        $k_o \leftarrow k_o \times \alpha_{k_o}$
    **end if**
    Update the knots $\psi_{i+1} \leftarrow \psi_i + \delta\psi_i$
    $iters \leftarrow iters + 1$
**end for**
```

for the quadrotor is to reach 10 meters in positive body x axis direction while avoiding obstacles. The optimization algorithm finds a trajectory for which the standard deviation ellipsoids in blue do not intersect the obstacles in red while achieving the final goal. Thus, following this open-loop trajectory approximately guarantees that the quadrotor can navigate safely around the obstacles.

4.1.4 Experiment Setup

The goal of the experiments is to demonstrate safe obstacle avoidance behavior for a quadrotor by following an open-loop trajectory computed using the NMPC optimization technique. The obstacle avoidance experiments are conducted on a real quadrotor in an outdoor scenario, subject to unknown external disturbances, and surrounded by virtual obstacles. The obstacle avoidance behavior has been demonstrated in two ways. First, multiple executions of open-loop trajectories have been performed to verify that the quadrotor model can predict the quadrotor pose well. Next, the quadrotor is set to avoid three consecutive virtual obstacles reactively. The obstacle avoidance experiments require several components such as an obstacle detection algorithm, a position controller, and an online parameter estimation framework to work together as discussed below.

4.1.4.1 Hardware

A research grade Matrice quadrotor made by DJI [50] is used in the experiments. The quadrotor is equipped with a Guidance stereo camera sensor [49] whose data is used to produce high-quality position and orientation measurements at 100 Hz. The DJI autopilot is used to achieve desired body Euler angles roll, pitch, yaw and desired thrust. Figure 4.1 shows the quadrotor setup connected with Guidance and DJI autopilot.

4.1.4.2 Obstacle Detection

A virtual camera image is rendered on a georeferenced virtual map using the Open Source 3D Graphics Engine library [165] to provide a depth map for estimating the distance to obstacles. The virtual camera avoids the issue of sensor uncertainty in detecting the obstacles and allows for safe experimental testing of the quadrotor in an outdoor scenario. Figure 4.5 shows an example virtual image rendered from the quadrotor's position. The obstacle position is determined from the rendered frame by segmenting the depth map into foreground and background based on the global velocity direction of the quadrotor. The pixels inside a tolerance cylinder of 0.5m radius around the

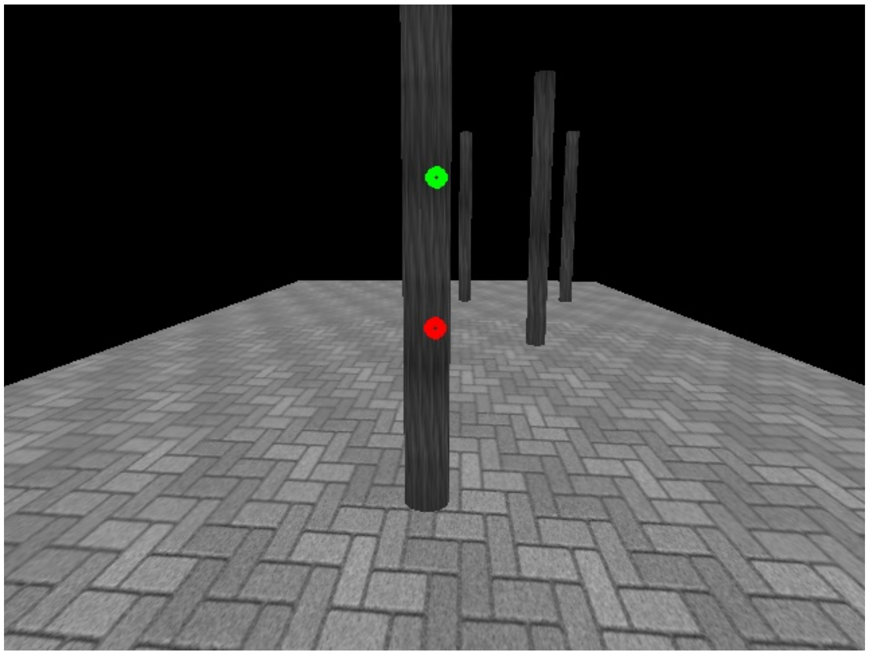


Figure 4.5: An onboard image rendered using a virtual camera. The green dot shows the mean of the closest 20 percent points in the foreground which represents the obstacle and the red dot shows the center of the image.

global velocity direction and within a maximum depth tolerance of 6m are considered to be foreground. The closest obstacle position is then specified to be the average of the closest 20 percent points in the foreground.

4.1.4.3 Online Parameter Estimation

The obstacle avoidance experiments are conducted in an outdoor environment with unknown external disturbances. Since the quadrotor battery keeps depreciating over time, same commanded thrust provides lower thrust as time progresses. These effects are compensated using MLE online parameter estimation of both system parameters and external disturbances. The initialization of MLE optimization requires a good prior on the system parameters.

Therefore, a manually flown ten-second trajectory is utilized in an initial optimization to produce a tight prior for the system parameters. The system parameters are further refined based on this prior using online optimization running at a frequency of 0.5Hz using the position and orientation measurements collected at 100 Hz. The covariance on the parameters is continuously tracked by the program and is used for fault detection in the case of bad parameter learning.

4.1.4.4 NMPC Trajectory Optimization

The NMPC optimization is performed when an obstacle is within a user specified tolerance distance of 3 meters. The goal of the optimization is to reach 3 meters behind the obstacle within the next two seconds while avoiding the obstacle and achieving a terminal velocity that is equal to the initial velocity. The two second trajectory is divided into 100 discrete segments (50 Hz) for optimization. In this experiment, the obstacles are avoided one at a time. Hence, a second order B-spline with four knots was sufficient to represent the control trajectory.

The computational resources available onboard are limited and only allow for a 10Hz frequency of the NMPC optimization loop. This restricts the frequency at which the optimal trajectory can be updated. Thus, the quadrotor is run open-loop using the roll, pitch, yaw and thrust inputs to autopilot until the MPC horizon time is complete.

4.1.4.5 Waypoint Tracker

A linear PID position controller is designed to track user-defined waypoints. The external force parameters and thrust gain obtained from MLE optimization are incorporated into the PID controller.

4.1.5 Results

4.1.5.1 Verification of quadrotor model

The ability of the quadrotor model to predict the quadrotor pose is demonstrated by executing open-loop trajectories multiple times to show that the quadrotor stays within the standard deviation funnel almost all the time. The trials require the quadrotor to fly at a speed of 3m/s and execute an open-loop NMPC trajectory to avoid a virtual obstacle assumed to be 3m in front of it. The data samples from the trials are segregated into a bar graph in Figure 4.6 based on the distance to the ellipsoid surface. Around 89.89% of the 2000 sample data points are within the ellipsoid as observed from the distance metric being negative in the bar graph. Under ideal conditions, if we computed the 2σ confidence around the state trajectories and assuming the state distribution is gaussian, the ellipsoids should contain 95% of the samples. The discrepancy between the experimental and theoretical sample percentages are probably because of the approximations induced by the simplified model considered for the quadrotor dynamics and using unscented transform to propagate the uncertainty. Despite the inaccuracies, the results suggest that by following the NMPC trajectory, the quadrotor is likely to avoid the obstacle with approximately high probability.

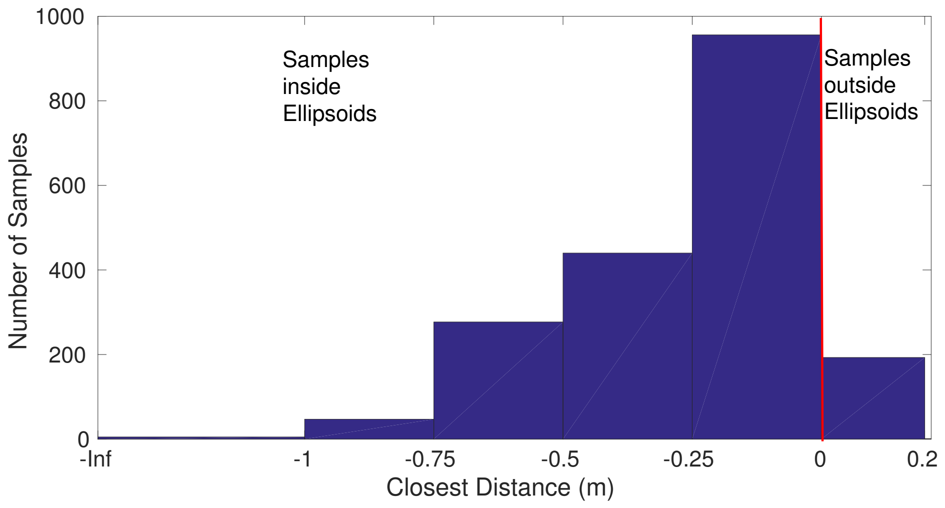


Figure 4.6: The bar graph shows the number of data samples at different distances to ellipsoid. Around 89.89% of the samples along the quadrotor trajectories are within the standard deviation ellipsoids and thus are safe for obstacle avoidance

Figure 4.7 shows the predicted and measured open-loop Euler angles and Figure 4.3 shows the predicted and measured position of the quadrotor along an experimental trajectory for a single obstacle avoidance trial. It can be observed that the predicted position stays within the standard deviation funnel and predicted Euler angles matched well with the measured Euler angles. These results indicate that the identified model approximates the quadrotor well, and NMPC trajectories are safe for avoiding a real obstacle.

4.1.5.2 Consecutive Virtual Obstacle Avoidance

In this experiment, the quadrotor is tasked to avoid a series of three virtual cylindrical obstacles 0.3 m in diameter and spaced 5m apart while following a series of waypoints at 3m/s. The quadrotor determines a obstacle avoidance trajectory when it sees a cylinder at 3 meters in front of it. The goal of the

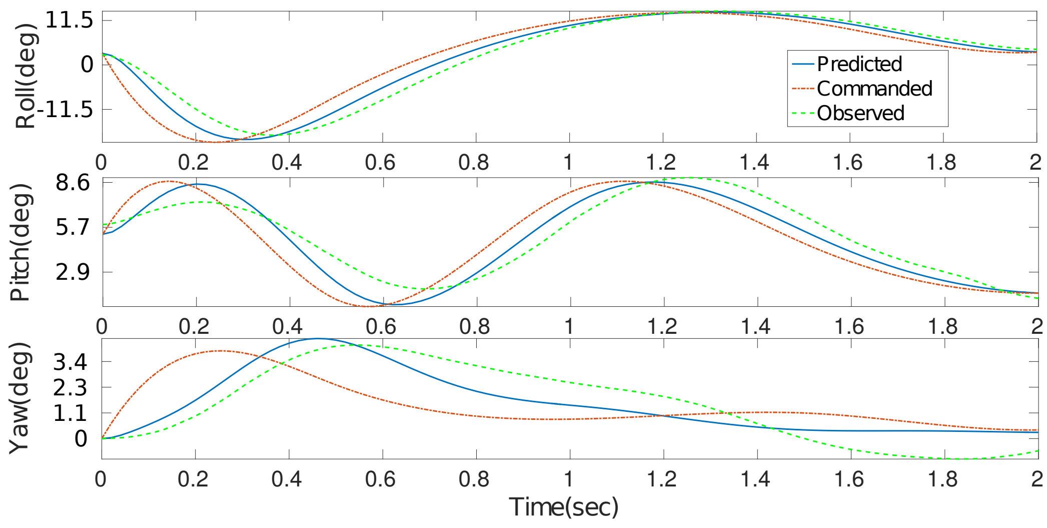


Figure 4.7: The graphs show the predicted, commanded and measured body Euler angles for a quadrotor during a single obstacle avoidance run. It can be observed the measured angles are very close to the predicted Euler angles.

NMPC optimization is to reach 3 meters beyond the obstacle within the next two seconds while maintaining the current quadrotor velocity. The combined subsystems of online parameter estimation, obstacle detection, NMPC trajectory optimization, and waypoint tracking are tested through this experiment. The upper half of Figure 4.8 shows the quadrotor positions in the virtual map overlayed with detected obstacles in red and computed NMPC trajectories with covariance ellipsoids in blue. The obstacles are inflated by 0.3m to include the envelope of the quadrotor. The series of pictures in the lower half of the figure show the quadrotor flying through the virtual obstacle course. The figures show that the quadrotor avoids the obstacles successfully while staying in the predicted standard deviation funnel when executing the NMPC trajectory.

The following steps are performed during an obstacle avoidance experiment:

1. Identify model parameters online by running MLE-based parameter estimation at 0.5Hz
2. Follow a waypoint reference trajectory using PID position controller
3. When an obstacle is detected in the path within a user specified tolerance, compute and execute NMPC trajectory
4. Resume following waypoint trajectory
5. A human operator takes over the controls if any of the above steps go wrong

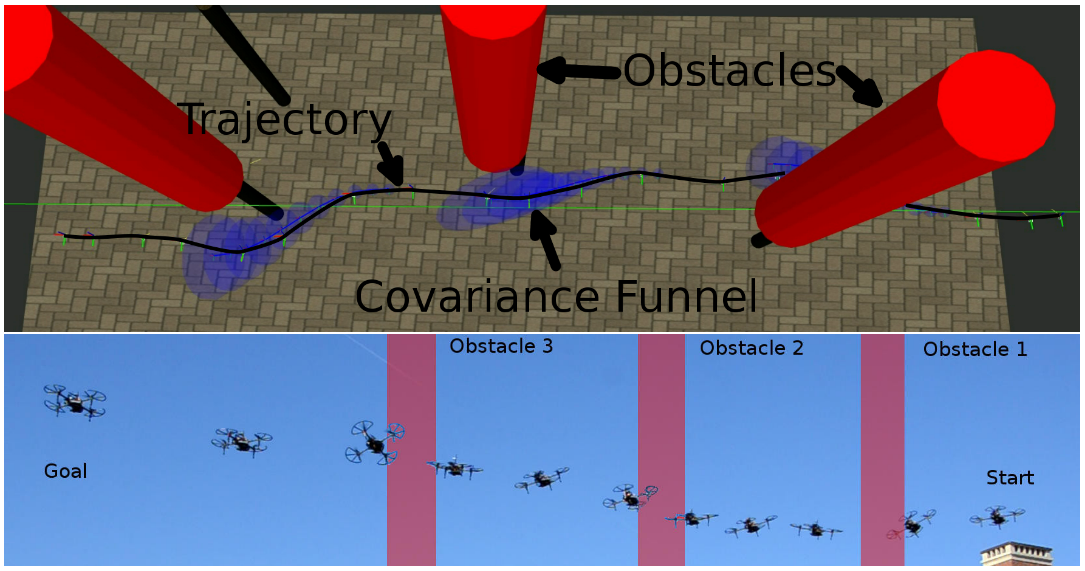


Figure 4.8: Pictures show a quadrotor moving at 3m/s evading three virtual cylinders by following an open-loop NMPC trajectory. The quadrotor stays within the funnel while avoiding the obstacles as shown in the figure in the top row. The figure in the bottom row shows the physical quadrotor flying through the obstacle course.

4.1.6 Conclusions

This section proposed an NMPC trajectory generation technique that combined online parameter estimation with uncertainty propagation to generate approximately safe obstacle avoidance trajectories. The ability of the nonlinear stochastic quadrotor model to predict the quadrotor state has been demonstrated through multiple open-loop trajectories. Further, the quadrotor is able to reactively avoid consecutive virtual obstacles while staying in the standard deviation funnel while following the NMPC trajectories.

4.1.7 Limitations

The current approach is limited in several ways. The adaptive NMPC approach combines a system identification module with a NMPC module to control the robot. Although each of the modules are stable by themselves the combined system might not be stable under certain circumstances since for general nonlinear systems we cannot separate optimal estimation and optimal control [95]. In our method, we try to get a good prior for system parameters by performing a manual data collection before starting the online system identification. We further monitor the stability of the online estimation procedure based on the estimated covariance of the parameters and decide if the combined system is stable or not.

The uncertainty propagation scheme employed in this chapter is approximate and is based on open-loop application of controls. In practice, we often use feedback controllers to stay close to the reference trajectory. Although unscented transform can be used to propagated closed-loop uncertainty it is still approximate. The following section will focus on finding a procedure to propagate uncertainty using a feedback controller with tighter guarantees.

4.2 Tube Nonlinear Model Predictive Control

4.2.1 Introduction

This section expands on the work from previous section 4.1 to consider propagating uncertainty in closed-loop for general robotic systems. We consider the problem of computing disturbance invariant sets for general nonlinear systems attached with arbitrary feedback controllers. To understand the problem, let us consider an agile aerial vehicle navigating autonomously to a goal in an obstacle-ridden environment subject to natural disturbances arising from wind or propeller downwash. This type of task is challenging since one has to account for the effect of disturbances on the system dynamics while planning a trajectory to avoid obstacles. Moreover, the disturbances are sometimes dependent on the state of the vehicle and, therefore, have a nonlinear effect on the propagation of the dynamics. For example, disturbances due to wall effects [164] scale based on the distance to wall. To account for the effects of uncertainty, current methods plan for a trajectory around the obstacles such that the obstacles do not intersect with the invariant set centered around the trajectory. The invariant set is defined as the region in state space in which the system is guaranteed to stay under the effect of disturbances for a given controller [142]. These methods usually depend on the structure of the dynamics or the controller and are not applicable to general nonlinear systems with nonlinear noise models. The goal of this work is, therefore, two-fold. We first propose a novel method to find approximate invariant sets for a general nonlinear dynamic system that can handle nonlinear noise models and which extends to higher dimensional systems. Next, we formulate a Nonlinear Model Predictive Control (NMPC) optimization that computes a nominal trajectory which avoids obstacles by relying on the computed invariant regions. Combining both the steps, we aim to tackle the obstacle avoidance problem for general nonlinear systems under external disturbances without any assumptions about the structure of the controller or the dynamics. For instance, one such obstacle avoiding trajectory generated for a simple wheeled vehicle with a nonlinear noise model is shown in Figure 4.9.

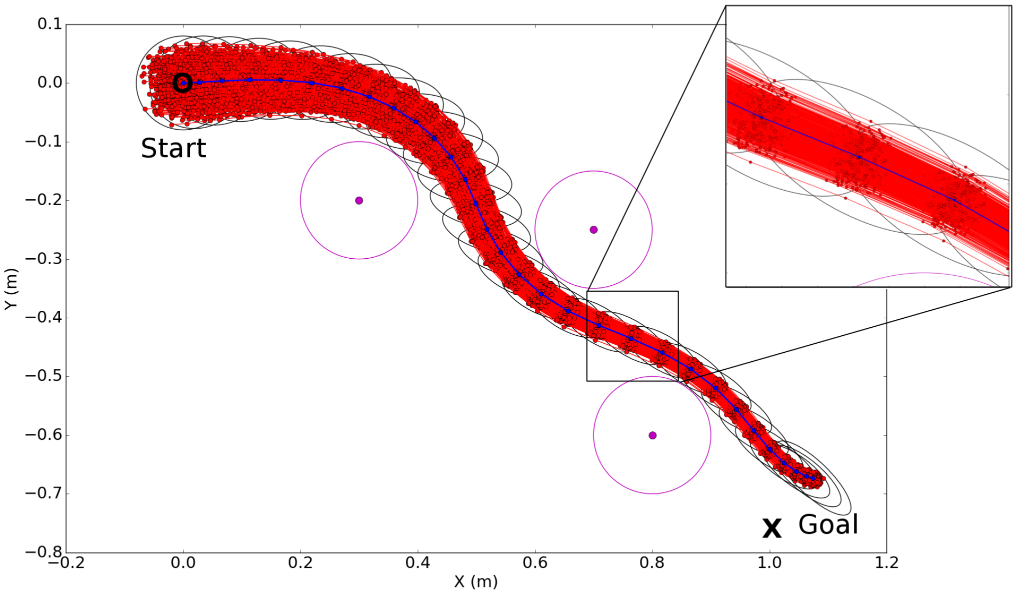


Figure 4.9: Unicycle model subject to external disturbances avoiding obstacles. The blue dots and lines are the nominal trajectory while the red dots and lines are sample trajectories propagated using the computed control law.

Related work

Disturbance-invariant sets provide one means of studying the effects of uncertainty on dynamical systems. These are regions in the state-space of a system that guarantee that if the system starts in the invariant set, then the system will lie in the invariant set under bounded disturbances. Conservative approximations for invariant sets have been computed for linear systems [191] and for piece-wise affine systems [192].

Tube-based Model Predictive Control (MPC) techniques have been developed that compute the invariant sets online for linear dynamics [147] and have also been extended to certain classes of nonlinear-discrete systems [193]. These techniques design controllers that provide "robust asymptotic stability" to the control invariant region.

Semi-definite optimization techniques [181], such as sum-of-squares (SOS) programming, search for polynomial functions that are positive and produced from a sum of squared monomials.Tobenkin, Manchester, and Tedrake [222] use SOS programming to compute funnels for nonlinear dynamics with a LQR controller. These funnels are time-varying state space regions such that if the system starts from the mouth of the funnel, it is guaranteed to stay in the funnel for all future time and eventually end up at the goal. By combining funnel computation with motion primitives, Majumdar and Tedrake [142] were able to compute robust funnels for a few primitives and perform online optimization on the order and duration of execution of each of the primitives to avoid obstacles.Steinhardt and Tedrake [214] explored finite-time verification for stochastic systems.

SOS programming has also been used to compute funnels for general Lipschitz nonlinear systems [232]. This approach computes the funnel around a nominal trajectory using invariant coordinates and optimizes only the nominal trajectory online. Unlike the previous method, this approach does not require recomputing funnels for every nominal trajectory since the funnel is specified in invariant coordinates around the nominal trajectory, and transformations to the nominal trajectory also transform the funnel around it. The main drawback with this approach is that the computation of the Lipschitz constant for a system is non-trivial [210]. Gao et al. applied this work to autonomous ground vehicles [59] and showed that tube MPC can avoid obstacles safely while following a desired trajectory.

Manchester et al. introduced Control Contraction Metrics (CCMs) as an alternative to Lyapunov functions to design globally stabilizing controllers for general nonlinear systems [143, 144]. Singh et al. used CCMs to compute the funnels for a nonlinear system [210]. In this approach, the funnel is computed in invariant coordinates and only the nominal trajectory is optimized online to perform obstacle avoidance. The success of this method relies on computing a valid CCM for the nonlinear system using SOS programming.

A different class of methods uses game-theory to find the worst possible noise for a given control [37, 215]. These methods usually contain a bi-level optimization [40] scheme in which the outer optimization searches for the controller that stabilizes the system while the inner-loop optimizes over the constraints to satisfy the process noise. The usual drawback with these methods is that they do not extend well to higher dimensional systems [210]. The approach presented here falls into this class. The algorithm presented here alleviates to some extent the curse of dimensionality by using a novel method to find the funnel and avoids the usual inner-outer optimization with a sequential optimization scheme that is shown to work well in the experiments conducted.

In addition to funnel methods, there are a class of techniques that treat states as a distribution and plan in the space of the distributions known as belief space [174, 182]. These can handle unbounded disturbances as long as the expected trajectory cost which they minimize is finite. Work done by Desaraju, Spitzer, and Michael [46] showed belief space planning for a small quadrotor platform. Chance constrained programming [34] is a further extension of these methods to handle probabilistic state constraints. In chance constrained programming, the obstacle constraints and other state constraints are respected only in probability. Although chance constrained programming is usually restricted to linear systems due to the difficulty in propagating multivariate distributions through nonlinear dynamics, there have been some attempts to extend these techniques to nonlinear systems [129, 54].

The methods discussed so far provide either deterministic or probabilistic guarantees for the stability of a controller under process noise. There also exists a class of risk-sensitive controllers [39] that provide some robustness to process noise without providing any guarantees on the performance of the controllers. For example, Manchester et al. optimized controllers to minimize their sensitivity to disturbances while not necessarily providing guarantees in terms of state-space funnels [145]. This method scaled well to higher dimensional systems such as robot arms and quadrotor models.

Existing challenges

In general, the aim of the methods discussed above is to stabilize a general nonlinear system to a goal state under bounded disturbances. They use state-space funnels to describe the possible trajectories the uncertain system might take for given a noise model. The funnel computation described in the methods discussed above are limited in several ways. Work done by Majumdar and Tedrake [142] requires the computation of a Lyapunov candidate using SOS programming and therefore is limited to polynomial dynamics. Similarly, funnel computation for Lipschitz nonlinear systems has been shown by Yu et al. [232], which requires computing the Lipschitz constant of the system. The work done by Singh et al. [210] also relies on SOS for computing a Control Contraction Metric to define the funnel. In addition, this method assumes an additive noise model which can be restrictive in practice. In addition, most methods discussed above presented results for systems with only a small number of dimensions and assumed linear noise models. For example, Singh et al. showed funnel computation for only a planar quadrotor, which is a six dimensional system [210] with crosswind disturbances. Work done by Majumdar and Tedrake [142] is among the few examples of high-dimensional funnel computation for a quadrotor, a 12-dimensional system.

Contributions

Our approach to finding the disturbance invariant set is to reduce the invariant set computation to a set of finite dimensional optimizations which provide an approximation to the invariant set, and a sequential NMPC is formulated to solve a boundary-value problem for avoiding obstacles using the approximate invariant sets computed. The NMPC consists of two optimizations running one after another. In the first optimization, the nominal trajectory is optimized to ensure the approximate invariant set around it avoids the obstacles. In the second optimization, the invariant set approximation is improved given the dynamics, the nominal trajectory, and the controller.

The algorithm, although providing only approximate guarantees, based on the experiments conducted, extends well for higher dimensional systems and can handle nonlinear noise models. For example, the quadrotor example shown in this work is a 14 dimensional system with a nonlinear backstepping controller. The presented approach does not rely on the specific structure of the dynamics, controller, or noise. Hence, it can be used to work with controllers without explicit stability certificates. For example, this method allows learning based controllers (e.g. Psaltis, Sideris, and Yamamura [185] and Garimella et al. [60]) to be implemented in practice by providing approximate guarantees using the invariant sets. Based on experiments performed on different systems, we show that the proposed method is able to compute good approximations of the invariant sets for high dimensional systems and effectively leverage them for robust obstacle avoidance.

4.2.2 Computing Invariant Funnels

4.2.2.1 Disturbance Invariant Sets

Consider a discrete nonlinear dynamic system with state $x \in \mathbb{R}^n$, control $u \in \mathbb{R}^m$, and a disturbance $w \in \mathbb{R}^n$. The dynamics can be written as

$$x_{i+1} = f(x_i, u_i, w_i) \tag{4.7}$$

where i denotes the discrete time index and the disturbance is assumed to be bounded (i.e. $\|w_i\| \leq \epsilon_i$). We assume a feedback controller ψ based on the current state x_i, a given goal state $\overline{x}_i$, and feed-forward controls $\overline{u}_i$ to compute the control u_i:

$$u_i = \psi(x_i, \overline{x}_i, \overline{u}_i). \tag{4.8}$$

The controller usually has additional parameters, such as feedback gains, which are assumed to be fixed and known in the rest of the section.

Assuming the state at step i is in some enclosing region P_i (i.e. $x_i \in P_i$), the disturbance invariant set at stage $i+1$ is defined as

$$P_{i+1} = \{x_{i+1} = f(x_i, \psi(x_i, \overline{x}_i, \overline{u}_i), w_i) \mid x_i \in P_i, \|w_i\| \leq \epsilon_i\}.$$

To author's knowledge, finding the smallest disturbance invariant set (in terms of volume) for a general nonlinear system with an arbitrary nonlinear controller cannot be solved in finite time and finite memory. The difficulty arises in searching for the boundary of the invariant set and representing the shape in a finite memory.

One solution to this problem [215] is to use a conservative estimate of P_i

denoted by C_i, i.e. $P_i \subseteq C_i$, and propagate C_i to C_{i+1} subject to the closed loop dynamics. Since the enclosing region C_i contains P_i, the enclosing region C_{i+1} also encloses P_{i+1}. Constraints, such as obstacles, can then be handled in a robust fashion by ensuring no overlap with the enclosing regions C_i. For the rest of the section, we choose the shape of C_i to be an ellipsoid with the same dimension as the state x_i. The choice of the family of enclosing region (e.g. ellipsoid, cuboid, sphere) affects the dilation between P_i and C_i and therefore how conservative the resulting obstacle avoidance trajectory will be. Furthermore, the estimate C_i becomes more and more conservative as the index i increases since the ellipsoid C_{i+1} is propagated from ellipsoid C_i instead of the actual region P_i. In spite of these drawbacks, propagating and reasoning in terms of conservative regions permits robust NMPC-based obstacle avoidance under disturbances.

To formulate the ellipsoid propagation problem, first we define the region of dynamics obtained by propagating the ellipsoid C_i through the dynamics as

$$\overline{P}_{i+1} = \{x = f(x_i, \psi(x_i, \overline{x}_i, \overline{u}_i), w_i) \mid x_i \in C_i, \|w_i\| \leq \epsilon_i\}.$$

Unlike, the region P_{i+1} which is propagated from P_i, the region $\overline{P}_{i+1}$ is propagated from the previous ellipsoid C_i (see Figure 4.10). Hence, the region $\overline{P}_{i+1}$ contains the region P_{i+1} (i.e. $P_{i+1} \subseteq \overline{P}_{i+1} \subseteq C_{i+1}$).

Finding the least conservative enclosing ellipsoid can be mathematically stated as

$$\min_{C_{i+1}} Vol(C_{i+1}) \;\text{ s.t } \overline{P}_{i+1} \subseteq C_{i+1}, \tag{4.9}$$

where $Vol(\cdot)$ denotes the volume of the ellipsoid. We simplify the above problem by constraining the centers of the ellipsoids to follow a nominal trajectory without disturbances. Assuming the center of C_i to be μ_i and the center of the C_{i+1} to be μ_{i+1}, the nominal trajectory dynamics are written as $\mu_{i+1} = f(\mu_i, u_i, 0)$. The control u_i used in the nominal trajectory dynamics is provided by the controller with the goal being the same as the current state, i.e $u_i = \psi(\mu_i, \mu_i, \overline{u}_i)$. The feed-forward terms in the controller move the nominal state μ_i to the next nominal state μ_{i+1}. The control for any other point in the invariant set is given by $u_i = \psi(x_i, \mu_i, \overline{u}_i)$.

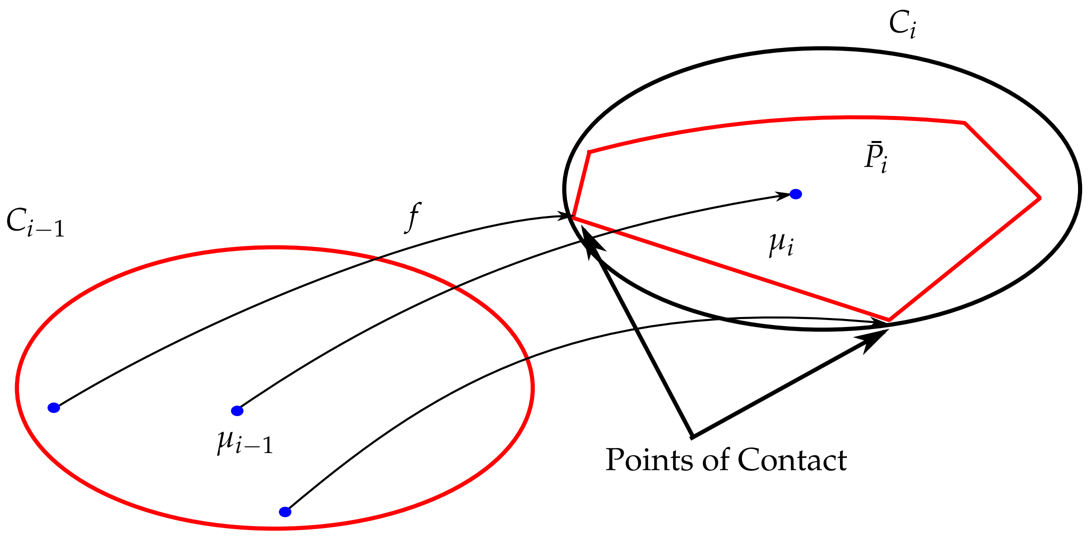


Figure 4.10: The region $\overline{P}_i$ is propagated from the previous ellipsoid C_{i-1} using the closed-loop dynamics f. The algorithm for computing the enclosing ellipsoids finds points in C_{i-1} (shown in blue) that when propagated become the points of contact between $\overline{P}_i$ and the enclosing ellipsoid C_i.

The ellipsoid optimization problem shown in eq (4.9) is intractable since the constraint $\overline{P}_{i+1} \subseteq C_{i+1}$ should be satisfied for all the disturbances $\|w_i\| \leq \epsilon_i$ and for all starting states $x_i \in C_i$. One approach to solve this problem is through SOS programming [181] under certain assumptions on the structure of the dynamics. In this work, we take a different approach and transform the above optimization problem into a finite dimensional optimization to find a feasible, but not necessarily the least conservative, ellipsoid.

4.2.2.2 Algorithm to Find Enclosing Ellipsoid

The algorithm has to find the radii $r_{i+1} \in \mathbb{R}^n_{>0}$ and principal axes $R_{i+1} \in SO(n)$ of the ellipsoid $C_{i+1} = \{r_{i+1}, R_{i+1}\}$ which encloses the propagated region $\overline{P}_{i+1}$ assuming the center of the ellipsoid is along some nominal trajectory. The procedure to find the ellipsoid consists of n successive optimization problems. Each optimization problem finds a principal axis and the ellipsoid radius along that principal axis. The first optimization simply searches for the farthest point from the center in the region $\overline{P}_{i+1}$:

$$r^2_{i+1,1} = \max_{x \in \overline{P}_{i+1}} (x - \mu_{i+1})^T (x - \mu_{i+1}). \tag{4.10}$$

The first radius and principal axis are chosen as $r_{i+1,1} = \|x_1 - \mu_{i+1}\|$ and $e_{i+1,1} = (1/r_{i+1,1})(x_1 - \mu_{i+1})$, respectively, where x_1 minimizes (4.10). Continuing, at step j of the optimization, we find a point $x_j \in \overline{P}_{i+1}$ that maximizes the radius along j-th principal axis assuming the point is on the boundary of the ellipsoid. The j-th radius $r_{i+1,j}$ and the principal axis $e_{i+1,j}$ of the ellipsoid assuming the point is on the boundary of the ellipsoid are given by

$$r_{i+1,j} = \frac{|x_{jj}|}{\sqrt{1 - \sum_{k=1}^{j-1} \left(\frac{x_{jk}}{r_{i+1,k}}\right)^2}}, \tag{4.11}$$

$$e_{i+1,j} = (x_j - \mu) - \sum_{k=1}^{j-1} e_{i+1,k} x_{jk}, \tag{4.12}$$

where x_{jk} is the projection of the point x_j along kth principal axis, i.e. $x_{jk} = (x_j - \mu_{i+1})^T e_{i+1,k}$. The vector $e_{i+1,j}$ is then normalized to ensure the jth principal axis is a unit vector. The j-th optimization problem can be formulated as

$$\max_{x_j \in \overline{P}_{i+1}} \; r^2_{i+1,j}, \;\; s.t. \;\; x_{jj} \neq 0. \tag{4.13}$$

The ellipsoid at C_{i+1} is given by

$$\begin{aligned} C_{i+1} = \{ \; & r_{i+1} = (r_{i+1,1}, \cdots, r_{i+1,n}), \\ & R_{i+1} = [e_{i+1,1} \mid e_{i+1,2} \mid \; \cdots \; \mid e_{i+1,n}] \; \}. \end{aligned} \tag{4.14}$$

Next, we prove that the ellipsoid generated by the above algorithm completely encloses the region $\overline{P}_{i+1}$ induced by the closed-loop dynamics, assuming the optimization in (4.13) reaches a global maximum. First, we prove the following lemma:

Lemma 4.2.1 (The cost function used in optimization (4.13) is bounded and $r_{i+1,j-1} \geq r_{i+1,j} \; \forall j \in \{2, \cdots, n\}$.)**.**

Proof. For $j = 1$, $r_{i+1,1}$ is given by (4.10). Assuming the optimization reaches global maximum, for any point x_2 in the region $\overline{P}_{i+1}$, using the definitions

$$x_{21} = (x_2 - \mu_{i+1})^T e_1, \qquad x_{22} = \|x_2 - x_{21}e_1\|,$$

we have $x_{21}^2 + x_{22}^2 = \|x_2 - \mu_{i+1}\|^2 \leq r_{i+1,1}^2$. Using the inequality, the cost function for $j = 2$ can be upper bounded as

$$r_{i+1,2}^2 = \frac{x_{22}^2}{1 - \left(\frac{x_{21}}{r_{i+1,1}}\right)^2} \leq \frac{r_{i+1,1}^2 - x_{11}^2}{1 - \left(\frac{x_{21}}{r_{i+1,1}}\right)^2} = r_{i+1,1}^2. \tag{4.15}$$

Next, for any $j > 1$, we already have r_{j-1} obtained by maximizing the cost function (4.13) at $j - 1$. Thus, for the point x_j in the region $\overline{P}_{i+1}$ which maximizes r_j, the following inequality holds

$$\frac{x_{jj}^2 + x_{j,j-1}^2}{1 - \sum_{k=1}^{j-2} \left(\frac{x_{jk}}{r_{i+1,k}}\right)^2} \leq r_{i+1,j-1}^2, \tag{4.16}$$

where x_{jj} is the residual left over after subtracting the components of $(x_j - \mu_{i+1})$ along the principal axes $\{e_1, e_2, \ldots, e_{j-1}\}$. This inequality can be transformed as

$$\frac{x_{jj}^2}{r_{i+1,j-1}^2} \leq 1 - \sum_{k=1}^{j-1} \left(\frac{x_{jk}}{r_{i+1,k}}\right)^2. \tag{4.17}$$

The cost function for the optimization at stage j can then be bounded as

$$r_{i+1,j}^2 = \frac{x_{jj}^2}{1 - \sum_{k=1}^{j-1} \left(\frac{x_{jk}}{r_{i+1,k}}\right)^2} \leq r_{i+1,j-1}^2. \tag{4.18}$$

Combining this inequality with the inequality for $j = 1$ given in (4.15), the lemma holds. □

Note that, we assume x_{jj} is not equal to zero while optimizing in (4.13). According to the inequality (4.17), for $x_{jj} \neq 0$, the right hand side of the inequality is greater than 0. Thus, the denominator of the $r_{i+1,j}$ is greater than zero. When x_{jj} is exactly zero, the cost function is undefined as both numerator and denominator are zero. Therefore, we avoid those points while minimizing the cost function.

Theorem 4.2.1 (The ellipsoid C_{i+1} as defined in (4.14) encloses the region $\overline{P}_{i+1}$.)**.**

Proof. Using the Lemma 4.2.1, we showed that the cost function is bounded. Now, consider the n-th optimization problem above which globally maximizes the last principal axis r_n. Since we find the maximum possible radius along the last principal axis, the radius computed using any other point $x_n \in \overline{P}_{i+1}$ is going to be less than $r_{i+1,n}$ i.e.

$$\frac{x_{nn}}{\sqrt{1 - \sum_{k=1}^{n-1} \left(\frac{x_{nk}}{r_{i+1,k}} \right)^2}} \leq r_{i+1,n} \tag{4.19}$$

$$\Rightarrow \sum_{k=1}^{n} \left(\frac{x_{nk}}{r_{i+1,k}} \right)^2 \leq 1 \tag{4.20}$$

where x_{nk} is the projection of the point x_n along the principal axis e_k (i.e. $x_{nk} = (x_n - \mu)^T e_k$). The inequality (4.20) then implies that C encloses the region $\overline{P}_{i+1}$. □

4.2.2.3 Simplification for cases near singularity

The cost function in (4.13) can become ill-defined when $x_{jj} \to 0$. Close to the singularities, the maximization of the cost function produces ellipsoids that are very conservative. Therefore, we develop an alternative optimization scheme that maximizes over x_{jj} instead of r_j when such singularities are encountered. The simplification improves the convergence of the optimization with the drawback that it produces an ellipsoid which only approximately encloses the propagated dynamics, i.e. the computed C_i may not completely enclose $\overline{P}_i$. The possible error between an ellipsoid computed by maximizing x_{jj} instead of r_j for a two-dimensional system is illustrated in Figure 4.11. The maximum error in the radius that can be encountered using this approximation is $\|r_j - r_{j+1}\|$ where r_j is computed by maximizing (4.13) and r_{j+1} is obtained using the proposed simplification.

The optimization scheme used in practice at stage j is given by

$$\min_{x_j} (x_{jj} - r_{max})^2, \;\; s.t \; x_j \in \overline{P}_{i+1}, \tag{4.21}$$

where the maximization in (4.13) is simplified and converted into a nonlinear least squares minimization where r_{max} is chosen such that $x_{jj} < r_{max}$. The least squares minimization is a local optimization technique, and hence the solutions obtained are not guaranteed to be global minima.

4.2.2.4 Summary

The ellipsoid computation scheme is summarized in algorithm 5. The computation scheme relies on finding a global minimum for the optimization

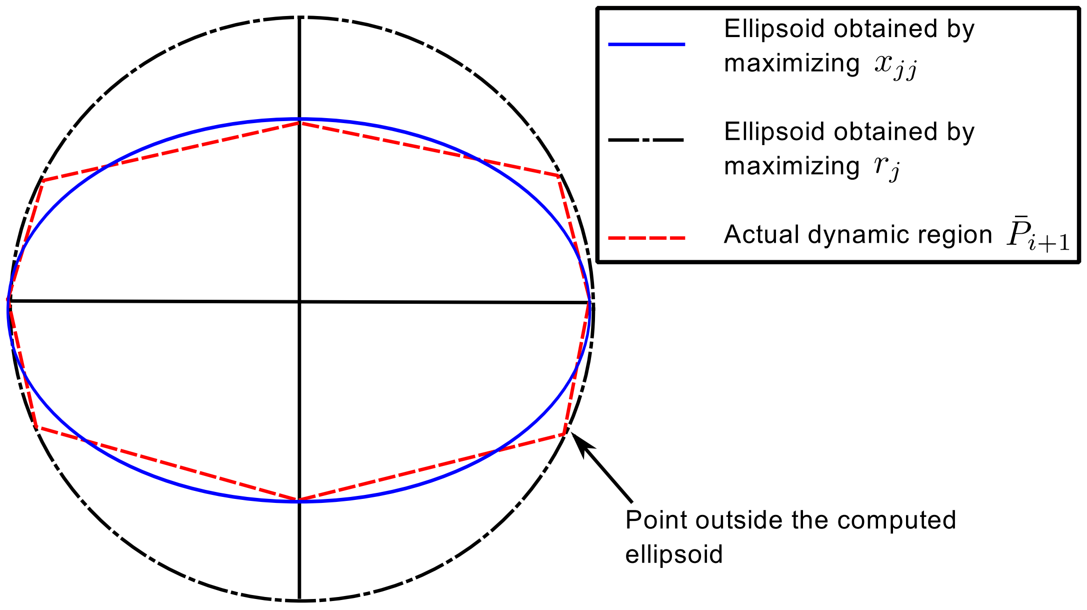

Ellipsoid obtained by maximizing x_{jj}
Ellipsoid obtained by maximizing r_j
Actual dynamic region $\bar{P}_{i+1}$
Point outside the computed ellipsoid

Figure 4.11: Error induced by simplification of the maximizing cost function. The ellipsoid obtained by maximizing only the projection x_{jj} shown in blue leads to a smaller ellipsoid, although it leaves out some parts of the dynamics region shown in red. The ellipsoid obtained by maximizing the minor axis r_j is shown in black and completely encloses the dynamics region.

in (4.21). In practice, we found that using a small time step for propagating the dynamics ensured that we were able to find a minimum close to the global minimum. The enclosing ellipsoid is completely defined by the starting points $p_{1:n}$, the noise terms $w_{1:n}$ and the center of the ellipsoid μ_i. Hence the ellipsoid propagation can be written as

$$C_{i+1} = g(p_{1:n}, w_{1:n}, \mu_i), \tag{4.22}$$

where the function g corresponds to the algorithm 5. The starting points and noise terms are used to formulate an approximate ellipsoid propagation scheme later.

Algorithm 5 Ellipsoid computation scheme

```
Given $C_0, N$
$j \leftarrow 0$
**for** $j < N$ **do**
    $\mu_{j+1} \leftarrow f(\mu_j, \mu_j, \bar{u}_j)$
    $i \leftarrow 0$
    **for** $i < n$ **do**
        Find $p_i$ in $C_i$ using eq (4.21)
        $i \leftarrow i + 1$
    **end for**
    Define $C_{i+1}$ in terms of $p_i, \mu_{j+1}$.
    $j \leftarrow j + 1$
**end for**
```

4.2.3 Robust obstacle avoidance

The robust obstacle avoidance problem can be stated as finding a nominal trajectory $\{\bar{x}_{1:N}\}$ and the enclosing ellipsoids centered around the nominal trajectory $C_{1:N}$ such that the terminal state reaches some goal state x_d and the enclosing regions do not intersect with the obstacles $o_{1:P}$. A system is steered toward the nominal trajectory using a controller with feed-forward control inputs i.e. $u_i = \psi(x_i, \bar{x}_i, \bar{u}_i)$. The obstacle avoidance problem can be mathematically stated as

$$\begin{aligned} &\min_{\bar{u}_{1:N}} L_f(\bar{x}_N, x_d) + \sum_{i=1}^{N} L_i(\bar{x}_i, \bar{u}_i) \\ &\bar{x}_{i+1} = f(\bar{x}_i, u_i, 0), \\ &C_{i+1} = f^{+}(C_i, \bar{u}_i, \bar{x}_i), \\ &dist(C_{i+1}, O_j) \geq 0, \end{aligned} \tag{4.23}$$

where f^+ propagates the disturbance invariant regions as described in section 4.2.2 and the *dist* function denotes the closest distance between the disturbance invariant region and the jth obstacle. The distance constraint should be satisfied for all enclosing ellipsoids C_i and all the obstacles O_j. The cost function L_f minimizes the distance between the nominal terminal state $\bar{x}_N$ and the desired state x_d. The trajectory cost L_i minimizes the feed-forward controller inputs and the nominal trajectory velocities. Thus, by minimizing the cost function, we find a nominal trajectory that reaches the terminal state while minimizing the control effort along the trajectory. The inputs to the optimization problem are the feed-forward control inputs $\bar{u}_i$ to the controller ψ.

The optimization problem described in (4.23) steers the nominal trajectory to ensure the enclosing ellipsoids do not intersect the obstacles. This optimization consists of a two-level inner and outer optimization. The inner optimization propagates the enclosing ellipsoids given the nominal state and feed-forward control inputs as described in section 4.2.2.2 while the outer optimization ensures the nominal trajectory is such that it minimizes the cost specified in (4.23) and the enclosing ellipsoids do not intersect with the obstacles.

This optimization is not practical since propagating enclosing ellipsoids for every single nominal state and control input is computationally expensive. Based on simulations conducted, this algorithm works for systems with two dimensional state but does not converge in a reasonable amount of time for systems with higher dimensions such as a unicycle or a quadrotor model.

4.2.3.1 Approximate Ellipsoid Propagation

The ellipsoid propagation described in section 4.2.2.2 performs n optimizations to find the points $p_{1:n}$ and the noise terms $w_{1:n}$ that when propagated determine the ellipsoid radii and principal axes completely as specified in (4.22). These points and noise terms are a function of the starting ellipsoid and the feed-forward control inputs. In the approximate propagation algorithm, we map the starting points from the input ellipsoid to a unit sphere as

$$e_j = diag(1/r_i) R_i^T (p_j - \mu_i), \text{ for } j \in \{1, \cdots, n\} \tag{4.24}$$

where $e_{1:n}$ are points inside a unit sphere and r_i, μ_i, R_i are the radius, center and principal axes of the input ellipsoid C_i. These mapped points are then assumed to be fixed even if the input ellipsoid and controller change. Therefore, given a new input ellipsoid $C_i' = \{r_i', \mu_i', R_i'\}$ and feed-forward control inputs $\bar{u}_i'$, the points on the unit sphere $e_{1:n}$ are projected back to the new input ellipsoid as

$$p_{1:n}' = \mu_i' + R_i' \, diag(r_i')(e_{1:n}). \tag{4.25}$$

The center of the ellipsoid at the next stage is found by propagating the center of the current ellipsoid using the feed-forward control inputs as

$$\mu_{i+1}' = f(\mu_i', \bar{u}_i', 0). \tag{4.26}$$

The radii and the principal axes of the ellipsoid are found by using the mapped points along with noise terms and the center of the ellipsoid in the ellipsoid propagation function specified in (4.22) as

$$C_{i+1} = g(p'_{1:n}, w_{1:n}, \mu_{i+1}). \tag{4.27}$$

4.2.3.2 Sequential NMPC

This approach consists of running two optimization steps repeatedly. The first step optimizes the feed-forward control terms assuming the ellipsoid is propagated using the approximate ellipsoid propagation algorithm. Given points $e_{1:n}$ inside a unit sphere, the feed-forward control optimization can be written as

$$\begin{aligned}
&\min_{\bar{u}_{1:N}} \; L_f(\bar{x}_N, x_d) + \sum_{i=1}^{N} L_i(\bar{x}_i, \bar{u}_i) \\
&\bar{x}_{i+1} = f(\bar{x}_i, u_i, 0), \\
&p_{i,j} = \bar{x}_{i,j} + R_i diag(r_i) e_{i,j} \\
&C_{i+1} = g(p_{i,1:n}, w_{i,1:n}, \bar{x}_{i+1}), \\
&dist(C_{i+1}, o_j) \geq 0.
\end{aligned} \tag{4.28}$$

To begin the optimization, the points $e_{1:n}$ are assumed to be columns of the identity matrix i.e., $\mathbb{I}_{n\times n} = [e_1, e_2, \cdots, e_n]$. The approximate ellipsoid propagation is used to propagate the ellipsoids along the nominal trajectory assuming the input points used for propagation are fixed. This avoids a costly inner loop optimization while incurring an error due to not updating the propagation points. In practice, as the controller converges to the optimal value, there is no change in the controller, and therefore, the error incurred due to not updating the propagation points is negligible.

In the second optimization step, the points $p_{1:n}$ used for ellipsoid propagation are optimized using (4.21) for the updated controller and nominal trajectory. This updates the worst possible propagated points and the noise at those points for a given controller and nominal trajectory. The combined algorithm therefore successively tries to steer the nominal trajectory and improve the enclosing ellipsoid approximation. As both the optimization steps converge, we obtain an optimal nominal trajectory with approximate disturbance invariant sets surrounding the trajectory. The pseudo-code for the sequential optimization scheme is summarized in algorithm 6

Algorithm 6 Sequential NMPC for Robust Obstacle avoidance

```
Given $C_0$, $x_0$, $x_d$, $L_i$, $L_f$, max_iters
Start with an initial guess of $e_{1:n}$ on unit sphere as $[e_1, e_2, \cdots, e_n] = \mathbb{I}_{n \times n}$
$i \leftarrow 0$
$iters \leftarrow 0$
for $iters$ <max_iters do
    $\min_{\bar{u}_{1:N}} \; L_f(\bar{x}_N, x_d) + \sum_{i=1}^{N} L_i(\bar{x}_i, \bar{u}_i)$,
    assuming approximate ellipsoid propagation to update $C_{1:N}$.
    for $i$ in 1 to $N$ do
        for $j$ in 1 to $n$ do
            Given $\bar{u}_i$,
            $\min_{p_{i,j}, w_{i,j}} (x_{jj} - r_{max})^2$,
            s.t $x_j = f(p_{i,j}, \bar{u}_i, w_{i,j})$,
            $p_{i,j} \in C_i, \|w_{i,j}\| \leq \epsilon_i$.
        end for
        $\mu_{i+1} \leftarrow f(\mu_i, \bar{u}_i, \mathbf{0})$.
        Use (4.11), (4.12) to find the enclosing ellipsoid $C_{i+1} = \{r_i, R_i\}$.
        Project: $e_{i,1:n} = diag(1/r_i) R_i^T (p_{i,1:n} - \mu_i)$.
    end for
end for
```

Computational Complexity

The computational complexity of the NMPC algorithm is evaluated in terms of the number of calls to the closed loop dynamics of the system. The sequential NMPC method contains two steps of optimization. The first step is a regular NMPC step with the additional complexity of propagating the ellipsoids. The ellipsoid propagation requires n calls to the dynamics to propagate the input points from one ellipsoid to the next. Thus, to propagate ellipsoids for an entire trajectory requires $O(Nn)$ calls to the dynamics. The usual NMPC approach without encoding sparsity takes $O(N^2m)$ calls for one step of optimization, assuming the gradient (of dimension Nm) is computed using a finite-difference approximation. Thus, the first optimization step of the proposed sequential NMPC approach takes $O(N^2(m+mn)) = O(N^2mn)$ function calls to the dynamics to compute gradients and update the feed-forward controls and the nominal trajectory. Ellipsoid propagation creates a linear dependence on the state dimension.

The second step of the sequential NMPC performs n least squares minimizations along each step of the trajectory. Each of those least squares minimizations makes $2n$ function calls to update the corresponding input point and the noise at that point. Therefore, the second step of the optimization takes $O(Nn^2)$ calls to update the ellipsoids along the entire trajectory. Therefore, a single step of the entire sequential NMPC optimization takes $O(N^2mn + Nn^2)$ function calls. The additional burden of using the sequential NMPC is a term that is only linear in the number of trajectory steps and is quadratic in state dimension.

4.2.4 Computational Results

We evaluate the robust obstacle avoidance procedure described in section 4.2.3 on two dynamical systems. The goal of the simulations is to show that the NMPC optimization can compute a feasible nominal trajectory that can avoid obstacles even when the system is subject to external disturbances. In addition, we show by sampling the disturbances and propagating several trajectories that most of the samples lie within the computed ellipsoids.

The first system is a dynamic unicycle model where the inputs to the model are the longitudinal acceleration a and the angular velocity ω of the vehicle. This system has a four dimensional state consisting of x and y positions, orientation θ and velocity v of the vehicle. To test the algorithm, we require the vehicle to drive to a goal position $(1m, -0.8m)$ with zero velocity and orientation. We added three obstacles located at $(0.8m, -0.15m)$, $(0.3m, -0.2m)$ and $(0.75m, -0.6m)$ with a radius of $0.1m$. The optimization has been performed using a quadratic cost with penalty on the control effort and terminal position.

Although the unicycle model lives on $SE(2)$ manifold, for computing funnels, we restrict our analysis to a single chart(restrict $|\theta| < \pi/2$) where the coordinates can be represented using a Euclidean manifold. The external disturbances $w \in \mathbb{R}^4$ are added into the model nonlinearly under the assumption that $\|w\| \leq 1$. The first two components of the noise correspond to the longitudinal and lateral noise respectively. The last two components of the noise correspond to the noise in angular velocity and acceleration. These components are scaled based on the velocity of the vehicle. The individual components of w are further scaled based on error magnitude $r_w \in \mathbb{R}^4$. The unicycle dynamics can be written as

$$\frac{d}{dt}\begin{bmatrix}x\\y\\\theta\\v\end{bmatrix} = \begin{bmatrix}v\cos(\theta)\\v\sin(\theta)\\\omega\\a\end{bmatrix} + \begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\\sin\theta & \cos\theta & 0 & 0\\0 & 0 & v & 0\\0 & 0 & 0 & v\end{bmatrix} diag(r_w)w. \tag{4.29}$$

A feedback linearizing controller is used to track the nominal trajectory. The controller is designed only for the nominal dynamics assuming the disturbances are zero. Thus, the controller is not Lyapunov stable under the disturbances. The noise added to the dynamics is nonlinear in terms of the state, which further complicates the ellipsoid computation.

The NMPC optimization formulated in (4.28) is applied to this model where the nominal trajectory acceleration is optimized to steer around the obstacles. Figure 4.9 shows a trajectory computed using the NMPC optimization. The nominal trajectory goes around the obstacles and the enclosing ellipsoids do not intersect with the obstacles. The noise entering the system is nonlinear and is higher in the longitudinal direction than in the lateral direction. This causes the ellipsoids to align with the trajectory heading as the system progresses. This example demonstrates that the algorithm can effectively handle nonlinear noise models.

Figure 4.12 shows that the ellipsoid constraint (4.20) for sampled trajectories evaluated on the ellipsoids projected onto the xy plane. The ellipsoid constraint being less than zero implies that the trajectories lie within the ellipsoids computed. Among 1000 sample trajectories evaluated only one sample trajectory violated the ellipsoid constraint. This suggests that the computed ellipsoids are able to capture the effect of disturbances on the unicycle dynamics. Thus, by following the nominal trajectory using the feedback linearizing controller, the unicycle can avoid the obstacles even under external disturbances.

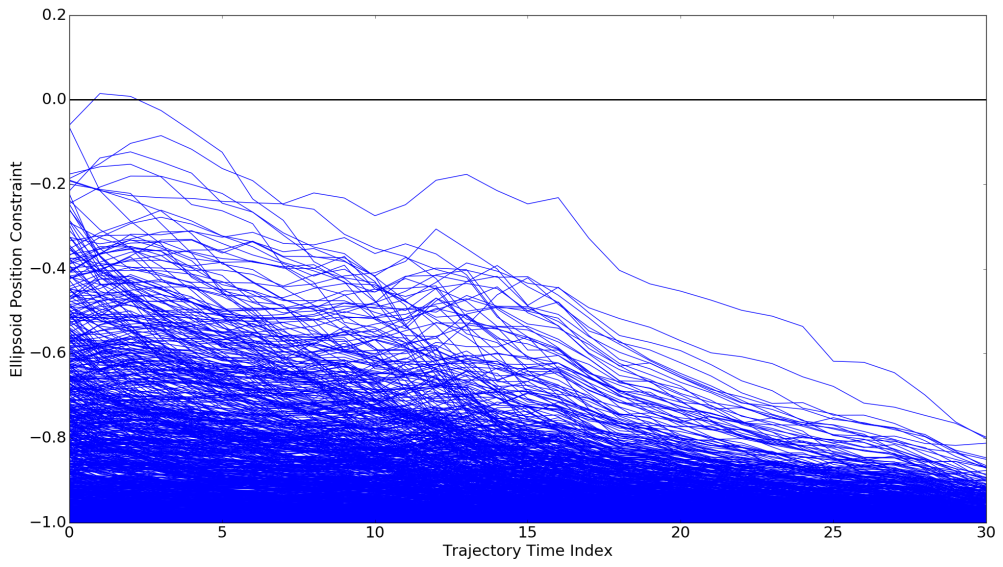


Figure 4.12: Ellipsoid constraint for ellipsoids projected along the xy plane evaluated for 1000 sample trajectories with uniformly randomly sampled disturbances and initial state. The constraint is less than zero if the sampled trajectory is within the invariant set.

The effect of the uncertainty on the planning process is explained in Figure 4.13. In this experiment, the process noise is increased in successive steps keeping the obstacles and the controller gains the same. Further, the NMPC optimization has been initialized using the same initial feed-forward controls in all the cases. As the process noise increases, the NMPC chooses a trajectory that is farther away from obstacles which increases the trajectory cost. It also shows that the above NMPC optimization succeeds in finding a safe trajectory even under increased process noise conditions.

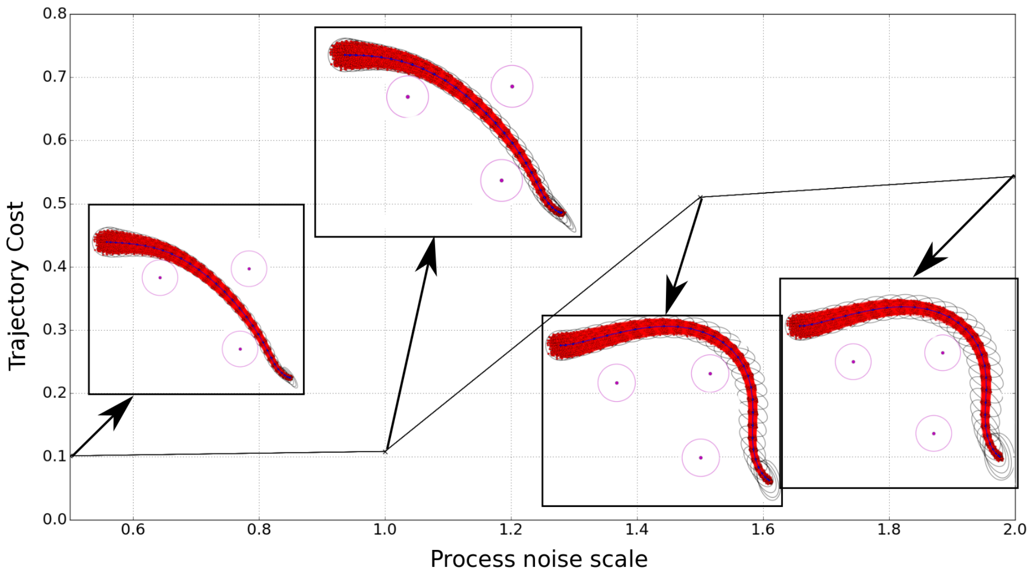


Figure 4.13: The effect of the process noise on the trajectory planning for a fixed set of controller gains and obstacles. As the process noise increases, the NMPC still succeeds in reaching the goal by taking a more conservative trajectory.

The second system considered is a quadrotor aerial vehicle whose dynamics is described in Kobilarov [108]. An additive noise is added to the nominal dynamics to test the disturbance invariance. The goal of the simulations is to fly a quadrotor from origin to the goal at $(0.8m, 0.8m, 0.8m)$ under disturbances and two obstacles located at $(0.5m, 0.5m, 0.5m)$ and $(0.2m, 0.2m, 0.4m)$ with a radius of $0.15m$. To test the effect of disturbances, the quadrotor is assumed to start from an initial ellipsoid of radii $(0.2, 0.1, 0.15)$ and initial Euler angles have an uncertainty of 0.05 radians and all other states have an uncertainty of 0.01 units. We also assume that the additive disturbances have a magnitude of 0.01 units during a time step of 0.1 seconds (for example the position noise will be 0.01 m in 0.1 seconds). The optimization is performed on a quadratic cost with a penalty on the terminal state and control effort during the trajectory.

Similar to the unicycle model, we restrict our analysis of the system to a single chart when computing the funnels around the quadrotor state(using Euler angle representation for orientation and restricting roll, pitch, yaw to be less than $pi/2$). A standard backstepping controller (such as the one described in Kobilarov [108]) is used for the nominal dynamics and while asymptotically stable in general, it loses strict stability guarantees under bounded additive noise. After performing a required *dynamic extension* of the model by adding the thrust and its time-derivative to the state, the state space becomes 14-dimensional. Figure 4.14 shows a nominal trajectory along with enclosing ellipsoids avoiding obstacles while reaching the goal. Figure 4.15 shows the ellipsoid constraint for 1000 sampled trajectories, where only two samples fall outside of the computed approximate invariant set. Figure 4.16 shows the 14 states along with the error bars along each axis that the states are expected to stay within. The error bars on the states converge to a small value suggesting that the controller is converging to a small region around the nominal trajectory. This example shows that the NMPC optimization can readily extend to higher dimensional systems such as an agile aerial vehicle in 3D.

4.2.5 Conclusion

This chapter developed an algorithm for computing approximate disturbance invariant sets around a nominal trajectory for general nonlinear systems governed by nonlinear feedback controllers under external disturbances. We applied the algorithm in simulations to demonstrate obstacle avoidance under model noise for two different dynamic systems: unicycle, quadrotor. Simulation results showed that the computed sets contain most of the samples and, therefore, are a good enough approximation to avoid obstacles under disturbances. This algorithm can be used to deploy experimental controllers on practical robotic systems with approximate guarantees on the safety of the closed loop system. In many robotic systems, the external disturbances have a nonlinear relationship with the state of the system. We demonstrate that the algorithm extends to nonlinear noise models in the case of a unicycle model. Future work will focus on reducing the dilation between the true region of the propagated dynamics and the approximate enclosing region by evaluating different families of regions other than ellipsoids. Currently, the external disturbances are assumed to be within a bounded sphere. Further research is necessary to model the external disturbances using high confidence bounds computed from sample trajectories and treat the obstacle avoidance constraints as a probabilistic constraints.

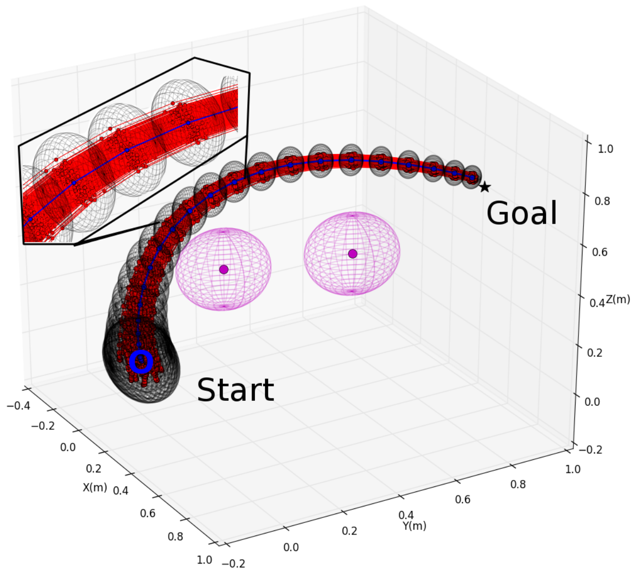


Figure 4.14: Quadrotor avoiding obstacles while reaching a goal. The nominal trajectory is blue while sample trajectories under the influence of disturbances are red. Obstacles are magenta, and the disturbance invariant sets are black ellipsoids surrounding the nominal trajectory. Note that the sample trajectories all fall within the computed approximate disturbance invariant sets.

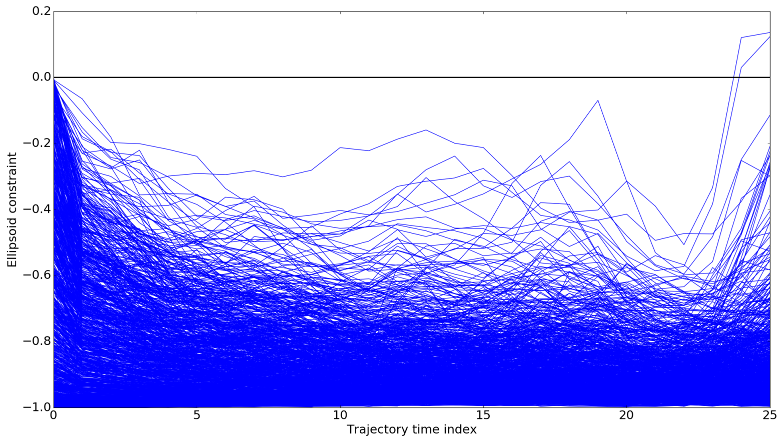


Figure 4.15: Ellipsoid constraint evaluated for 1000 sample trajectories with uniformly randomly sampled initial state and disturbances along the trajectory. The constraint is less than zero if the sampled trajectory is within the ellipsoid.

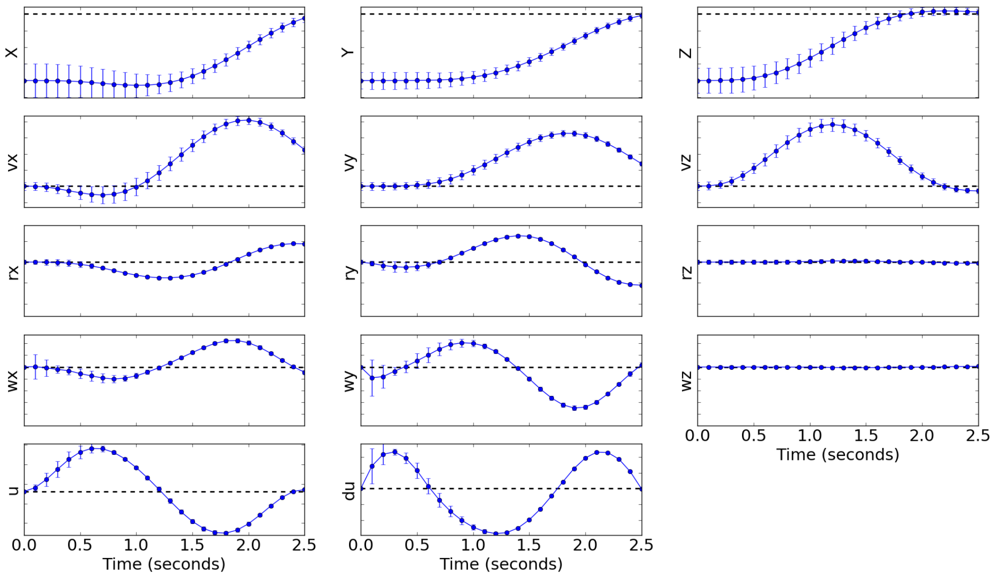


Figure 4.16: Quadrotor states along the nominal trajectory. Error bars denote the extent of the disturbance invariant set about the nominal trajectory. The black dashed line shows the desired state at the goal. The variables v, ω, r, u, and du are the velocity, angular velocity, RPY rotation, thrust, and thrust time derivative, respectively.

Chapter 5

NMPC using Recurrent Neural Network (RNN) models

The algorithms presented in previous chapters 3, 4 relied on a domain expert to design an accurate dynamic model of the robotic system. This chapter attempts to eliminate the dependence on the domain expert for designing a dynamic model for controlling a robot using NMPC. Using domain knowledge is sometimes restricting in designing a good dynamic model for the robot. For example, modeling the road wheel interactions for a passenger car requires understanding several different forces such as friction, damping, inertia in addition to the power train dynamics. We propose a RNN architecture that can learn robot dynamics by utilizing minimal prior knowledge about the robot. By using a data-driven model such as an RNN, we can design accurate dynamic models when the required domain knowledge is not available or is hard to obtain. We explore the effect of using limited prior knowledge about the robotic system on the size of the network and generalization error. We further combine the RNN with NMPC framework to control a general robot with minimal domain knowledge. This algorithm can then be applied to general robotic systems where a dynamic model is designed by exploring the system state space and combining the dynamic model with NMPC will provide us a way to drive the system to a goal state or follow a reference trajectory. We demonstrate the control algorithm on two robotic systems: aerial manipulation and ground vehicle.

5.1 Neural Network Modeling for steering control of an autonomous vehicle

5.1.1 Introduction

This section considers the dynamical modeling of steering systems in passenger vehicles and the use of derived models for steering control, as a basic building block for autonomous driving. We propose an architecture consisting of a nominal physics-based model that is embedded as a block inside a data-driven recurrent neural network (RNN). This RNN model serves as a transition function for model-predictive-control (MPC) to achieve desired steering behavior. The approach is suitable for systems with known electromechanical specifications but also for black-box (e.g. third-party OEM) vehicle systems with unknown characteristics. We show that this method achieves marked improvement over traditional feed-forward techniques and study the effects of different strategies for combining RNNs with a simple physics-based model.

The steering system can nominally be modeled based on known physics laws from first principles. The steering dynamics are usually modeled as a second-order forced dynamical system [200] with torque inputs induced by the steering actuator and tire forces [154]. For modern steering systems, a power steering motor is used as the steering actuator and is governed by a steering controller. In this work, we assume that no exact knowledge of the internal controller logic is available and the first-principles approach is thus a very coarse approximation to the actual behavior.

Unlike the first principles models, neural network models do not depend on strict physics-based assumptions and instead can be derived solely from experimental input-output data [183]. Such models can better capture complex actuator nonlinearities and delays and can thus provide higher predictive accuracy. Neural network models also require less domain specific knowledge [128]. A type of neural network model known as Recurrent Neural Network (RNN) has been traditionally used to model temporal dynamics in language models and handwriting recognition [88]. Early work showed the ability of RNN models to approximate nonlinear dynamic systems [41, 233]. Psichogios and Ungar [186] further showed that combining a first principles model with an RNN can approximate the nonlinear dynamics using a smaller training sample size and can extrapolate better to unseen samples. Several industrial control applications of RNN models have also been demonstrated such as controlling a steel prickling process [102], yeast drying process [234], and temperature control of a variable-frequency oil cooling machine [135]. Recently, RNN models have been employed in Model Predictive Control (MPC) framework to produce a controller for robot manipulation [83], using deep layers [172] to model the robot dynamics and controls cutting task accurately.

A neural network approach can be used in conjunction with control policy learning [175]. Deep neural networks have been proposed to learn a control policy based on experience. In a reinforcement learning setting, the control policy tries to maximize a reward function by balancing exploration and exploitation of the dynamics of the system [156]. Guided Policy Search Methods have been proposed to improve the convergence of the reinforcement learning methods [235]. While these techniques are very general and could work directly with the physical system, our present work focuses on first learning an accurate and robust dynamics model only, which can then be rigorously validated and used for traditional model-based control.

RNN-based models have been employed for vehicle controls in many simulation studies [52, 187, 69, 122]. Rivals et al. [195] experimentally demonstrated neural network based lateral control of a four-wheel-drive car. Apart from the model-based control schemes, vision based neural network control policies have also been developed [176, 35]. The control policies provide actuator commands to the vehicle based on the camera input obtained from the dashboard. However, these control policies rely heavily on the trained vision data and when presented with new examples, can perform unexpectedly. While the approach of mapping from sensory inputs to actions directly holds promise, for safety considerations we consider a more traditional multi-layered model-based approach consisting of high-level planner providing reference trajectories that are tracked by a low-level controller. This decomposition allows for individual verification of each layer on a wide range of datasets.

Contributions. In this work, a model based control strategy is developed for the steering dynamics of an automotive vehicle. An RNN model has been used to model the steering dynamics of the vehicle. The model is augmented with known physics-based transition blocks to improve its predictive capacity. The learned model is then employed in an MPC framework to produce the feed-forward control torques for a low-level embedded steering PID controller. Experiments on Toyota Highlander vehicles have been conducted to compare the performance of the steering tracking behavior using feedforward controls from different models. The limitations and advantages of several configurations of the proposed architecture are examined.

5.1.2 Control Architecture

Controlling the steering system on a Toyota Highlander requires applying torque on the steering wheel which is then amplified through a power steering system and applied to the front wheels. We automate this system by applying the input torques as a control signal to the power steering module. The goal of the steering controller is to track a reference steering trajectory obtained from a high-level planner as shown in Fig 5.1. The usage of a layered structure separates the vehicle behavior from low-level trajectory tracking. Following the control structure, the high-level planner produces a trajectory that is consistent with obstacles and road rules. The MPC trajectory tracking then produces a steering reference trajectory necessary to achieve the high-level planner trajectory. Finally, the low-level steering PID controller computes the actuator steering torque input required to track the steering trajectory.

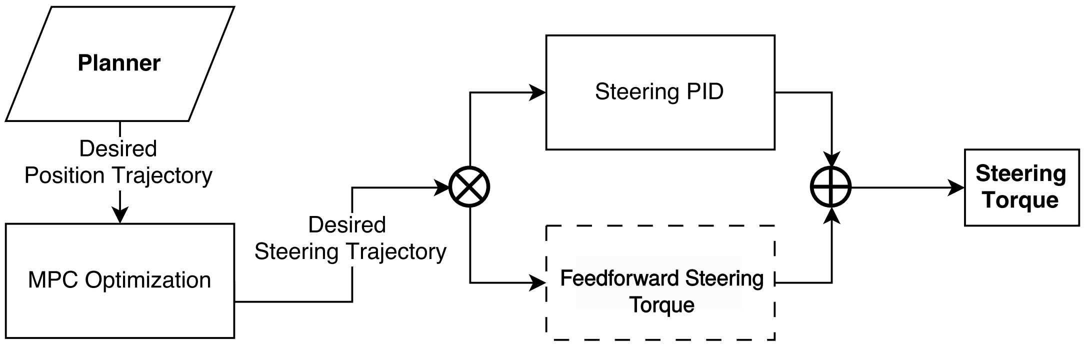

Planner
Desired Position Trajectory
MPC Optimization
Desired Steering Trajectory
Steering PID
Feedforward Steering Torque
Steering Torque

Figure 5.1: Control architecture showing how a desired trajectory is converted to steering actuator commands. The focus of the paper is to develop a feedforward steering torque to improve steering controller.

The focus of this work is to develop an appropriate feedforward steering torque input to improve the steering PID controller. The steering PID controller tracks a reference steering trajectory (δ_r, $\dot{\delta}_r$), by commanding the steering torque τ_{cmd} as

$$\tau_{cmd} = -k_p(\delta - \delta_r) - k_d(\dot{\delta} - \dot{\delta}_r) - k_i \int (\delta - \tilde{\delta})dt + \tau_{ff}.$$

Figure 5.2 shows a sample trial of following a steering reference trajectory using a tuned PID controller. The figure highlights the different component outputs of the PID controller as compared to the net output of the PID controller. It can be observed that the bulk of the control input during the tracking is provided by the integrator alone which leads to poor tracking performance. Tracking performance in theory could be improved by increasing PID gains, but this can lead to unstable behavior, especially given time delays inherent in the system. Using lower gains ensures stability but allows error to accumulate over time, which is then compensated through the integrator. By using an accurate feedforward steering torque τ_{ff} we expect to improve the system response and reduce tracking error without increasing PID gains. The models necessary for generating the feedforward steering torque are investigated next.

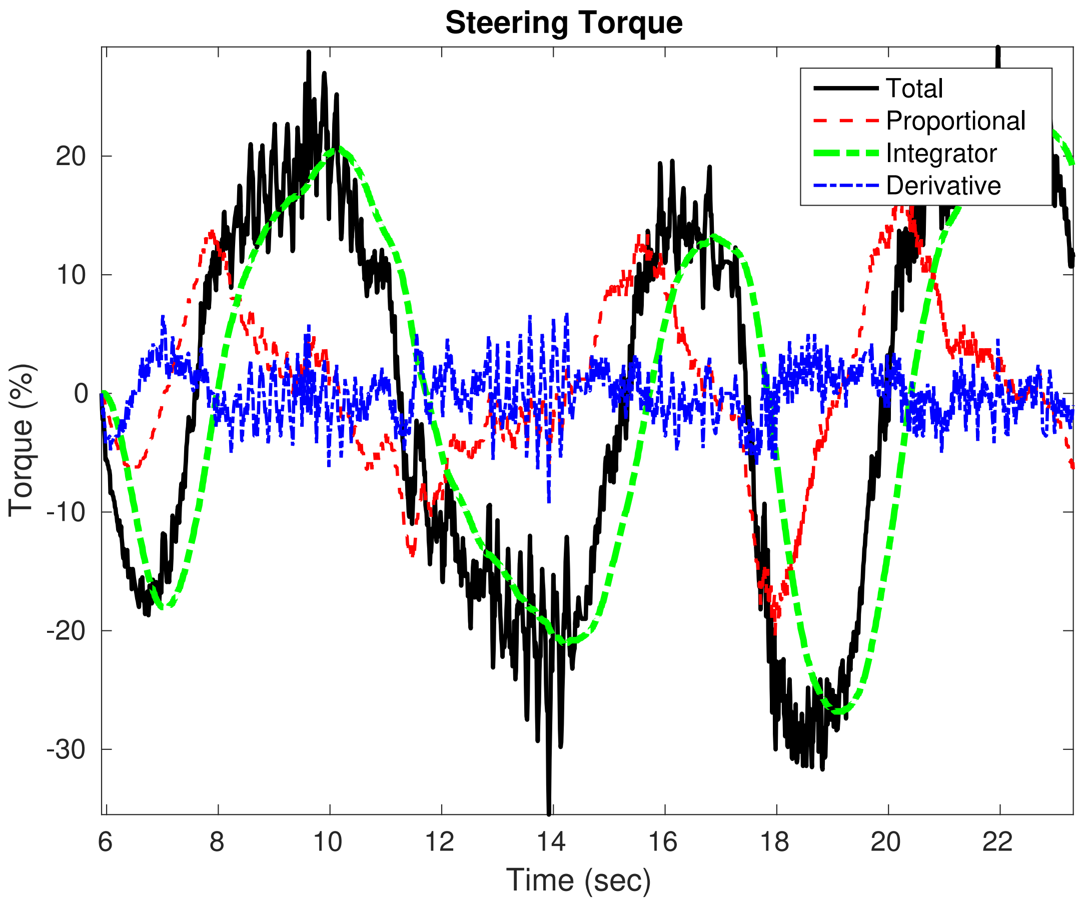


Figure 5.2: Steering torque generated by a PID controller during a trajectory tracking experiment. The majority of the torque output is provided by the integrator indicating that the steering system is highly nonlinear.

5.1.3 Modeling Steering Dynamics

The steering model predicts the steering angle δ and the steering rate $\dot{\delta}$ based on the applied steering torque τ_s. Steering dynamics are highly coupled with the lateral dynamics of the vehicle, which is in turn depends on the longitudinal velocity v_x, which we regard as a recorded input. The lateral states, lateral velocity v_y and yaw rate $\dot{\phi}$, are therefore included as a part of our steering dynamics formulation. The complete state of the steering dynamics is $x = [\delta, \dot{\delta}, \dot{\phi}, v_y]^T$ and controls by $u = [\tau_s, v_x]^T$. The stacked states and controls are represented by $z = [x^T, u^T]^T$.

The discrete steering dynamics evolve according to the discrete-time model $x_{i+1} = f(z_i)$ where i defines the time index along the trajectory. The discrete steering dynamics predicts the next state x_{i+1} given the current state x_i and control u_i. The state transition function f is an unknown nonlinear function of the previous state and controls. It can be derived from a first principles approach or using a neural network.

5.1.3.1 First principles model

The first principles model is a second order model that integrates the net steering torque to obtain the steering angle and its rate. The net steering torque is given by a combination of the steering system dynamics, road wheel interactions, and power steering torque. The steering system dynamics consists of Coulomb friction from the steering rack, jacking torque due to camber angle, and damping caused by rigid body dynamics [121]. The self-aligning torque is produced due to tire deformation when the steering wheel moves against the tire thread. At small slip angles, the self-aligning torque is proportional to the lateral force F_{yf} applied to the front tire. The resulting first principles model is

$$\ddot{\delta} = \underbrace{k_p\delta}_{\text{jacking}} + \underbrace{k_d\dot{\delta}}_{\text{damping}} + \underbrace{k_aF_{yf}}_{\text{Aligning}} + \underbrace{k_c\text{sgn}(\dot{\delta})}_{\text{Coulomb}} + \underbrace{g(\tau_s, v_x)\tau_s}_{\text{Power Steering}},$$

$$\text{sgn}(\dot{\delta}) = \begin{cases} 1 & \text{if } \dot{\delta} > 0 \\ -1 & \text{otherwise} \end{cases}.$$

A tire model specifies the lateral force applied to the front tire [154]. The lateral velocity and the yaw rate of the car are propagated using the bicycle model [200].

The power steering system applies an actuation torque based on the torque sensor input τ_s and longitudinal velocity v_x. The net torque from power steering system is modeled as a nonlinear gain $g(\tau_s, v_x)$ on the torque sensor input.

$$g(\tau_s, v_x) = k_1 g_1(\tau_s) g_2(v),$$

$$g_1(\tau_s) = \left[1 - e^{-\left(\frac{\tau_s - k_2}{k_3}\right)^2}\right],$$

$$g_2(v) = \left[k_4 + (1 - k_4)e^{-v_x/k_5}\right].$$

The power steering gain $g(\tau_s, v_x)$ is decomposed into two multiplicative gains. The first gain is dependent on driver input. It increases with the driver input and saturates to a constant value k_1 as driver input becomes large. The second gain decreases with vehicle velocity until it saturates to a constant value k_4. The form of the gain function has been chosen based on observed experimental data between driver input torque and assist torque.

The power steering dynamics presented here is an approximation based on the graphs shown in Aly et al. [10], since the true dynamics is unknown. The unknown parameters in the first principles model are obtained using standard least-squares regression based on the error between predicted and measured steering angle and steering rate.

5.1.3.2 Neural Network Model

The neural network model approximates the nonlinear function $x_{i+1} = f(z_i)$ to predict the discrete-time steering dynamics. The model consists of several units of neural network blocks stacked in time. Each neural network block is divided into a known physics function and a stack of fully connected layers. Figure 5.3 shows a block diagram of a neural network block. The current inputs and previous outputs are stacked together and fed into the fully connected layers and the physics function. The output of these layers is combined to predict the state at the next step. These predicted variables together with new control inputs are fed back into the network to continue prediction for the next step.

The overall discrete dynamics for a neural network model can be written as $f(z) = f_{ph}(z) + f_{nn}(z)$, where f_{ph} denotes the physics function and f_{nn} represents the neural network layers. The physics function incorporates basic domain knowledge by predicting the steering angle as an integral of the steering rate and the yaw rate based on the no-slip condition for a kinematic

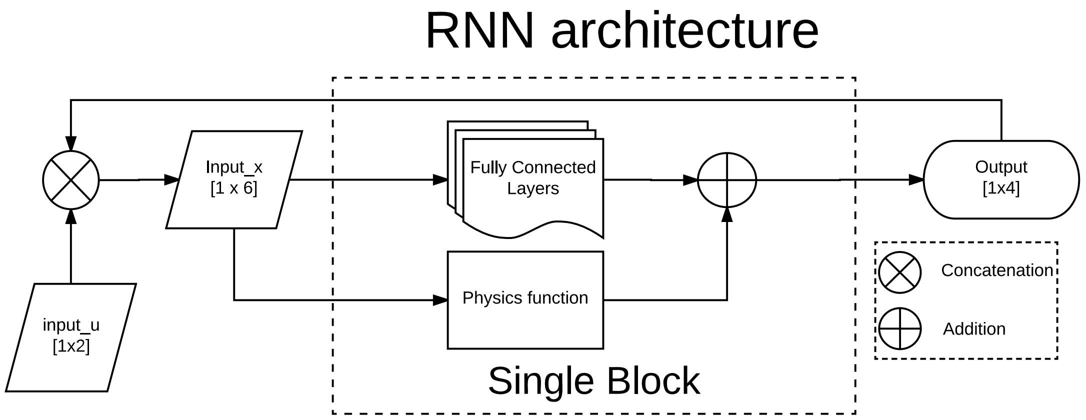


Figure 5.3: The neural network architecture used for learning lateral dynamics of a car model.

car model. The physics function can be mathematically stated as

$$f_{ph}(z_i) = \begin{bmatrix} \delta_i + k_1 \dot{\delta}_i \\ 0 \\ v_{xi} \tan(k_2 \delta_i) \\ 0 \end{bmatrix},$$

where the unknown parameters are the time-constant for integration k_1 and the inverse of the wheelbase of the car k_2. Similarly, the neural network layers can be expressed as

$$f_{nn}(z_i) = g_n\left(g_{n-1}, \left(g_{n-2}\left(\cdots g_0\left(z_i\right)\right)\right)\right),$$

$$g_i(x) := \sigma(W_i x + b_i), \quad i = 0, \ldots, n-1,$$

$$g_n(x) := W_n x + b_n.$$

The function $g_i(\cdot)$ defines a single fully connected layer for the neural network with the parameters W_i, b_i. The function $\sigma(\cdot)$ denotes the activation function used in the network. The activation function is chosen to be the hyperbolic tangent function. The last layer $g_n(\cdot)$ is specified as a unit activation function since the neural network is used in a regression problem.

The neural network layers predict the residual dynamics not modeled by the physics function. The goal of the neural network layers, therefore, is to capture the effects of road-tire interactions, power steering logic, and steering system dynamics. The addition of physics function improves the gradient flow of the network thereby providing better prediction performance as observed from training performance results in Fig. 5.4. The increase in depth of the fully connected layers increases the ability of the RNN to model higher nonlinear models. At the same time, it is also harder to train deeper neural network models due to the vanishing gradient problem [18]. In this work, the depth of the neural network layers has been experimentally determined to capture the unknown dynamics well.

Training

The neural network weights and the physics function parameters are optimized using time series data collected from driving the vehicle along variable curvature paths with different desired longitudinal velocities. The steering dynamics are controlled using a preliminary Proportional Integral Derivative (PID) controller during the data collection phase.

The collected time series data is divided into fixed time horizon segments to train the neural network model. The loss function during neural network training is set to minimize the difference between propagated states and measured states for each of the fixed time horizon segments. The loss function also adds an L2 regularization with a user-selected gain c_r to avoid overfitting of the parameters. The goal of the training phase is to find the optimal parameters θ^* that minimize the training cost as

$$\theta^* = \arg\min_{\theta} \sum_{j=1}^{m} C(\tilde{x}_{0:N}, u_{0:N-1}, \theta),$$

$$C(\tilde{x}_{0:N}, u_{0:N-1}, \theta) = \sum_{i=1}^{n} (\tilde{x}_i - x_i)^T P (\tilde{x}_i - x_i) + c_r \theta^T \theta,$$

$$\text{s.t} \quad x_{i+1} = f_{ph}(z_i, \theta) + f_{nn}(z_i, \theta),$$

$$z_i = [x_i, u_i], x_0 = \tilde{x}_0,$$

$$\theta = [W_0, b_0, W_1, b_1, \cdots, W_n, b_n, k_1, k_2],$$

where θ denotes the weights of the network along with the unknown physics parameters.

The cost function C measures the deviation of the propagated states x_i from measured states $\tilde{x}_i$ for m sample trajectories. The state deviations are weighed using a diagonal matrix P to enforce a uniform scale across the deviations. The matrix P is usually chosen as the inverse covariance of the sensor measurements.

The propagation of steering dynamics results in the propagated states significantly diverging from the measured states for random initialization of parameters. The optimization is thus performed in two stages: first by limiting the propagation to a single step and initializing the parameters obtained from the previous stage and optimizing over the entire trajectory segment. This two-stage optimization proposed by Lenz et al. [83] has been shown to perform better than random initialization of parameters.

Implementation

The RNN is coded as a computational graph in TensorFlow [218] package. The training data is provided by 20,000 trajectory segments with a 0.5-second horizon which corresponds to approximately 150 hours of driving data. The neural network has been trained with two fully connected layers. The optimization has been performed using mini-batch gradient descent with a batch size of 200 samples.

The optimal parameters are used to verify the performance of the model on a test data set of 5,000 trajectory segments. The RNN model is used to propagate the lateral dynamics using the initial state and controls along the trajectory. Figure 5.4 shows the RMS error between the propagated and measured handwheel angle along the trajectory using different trained models. The addition of a second layer to the RNN along with physics function improves the performance of prediction significantly. Increasing the RNN layers further does not result in an improvement of performance. Hence the depth of neural network is limited to two layers.

5.1.4 Steering Control

The steering torque used to track a steering reference trajectory consists of a PID component and a feedforward component. The steering reference

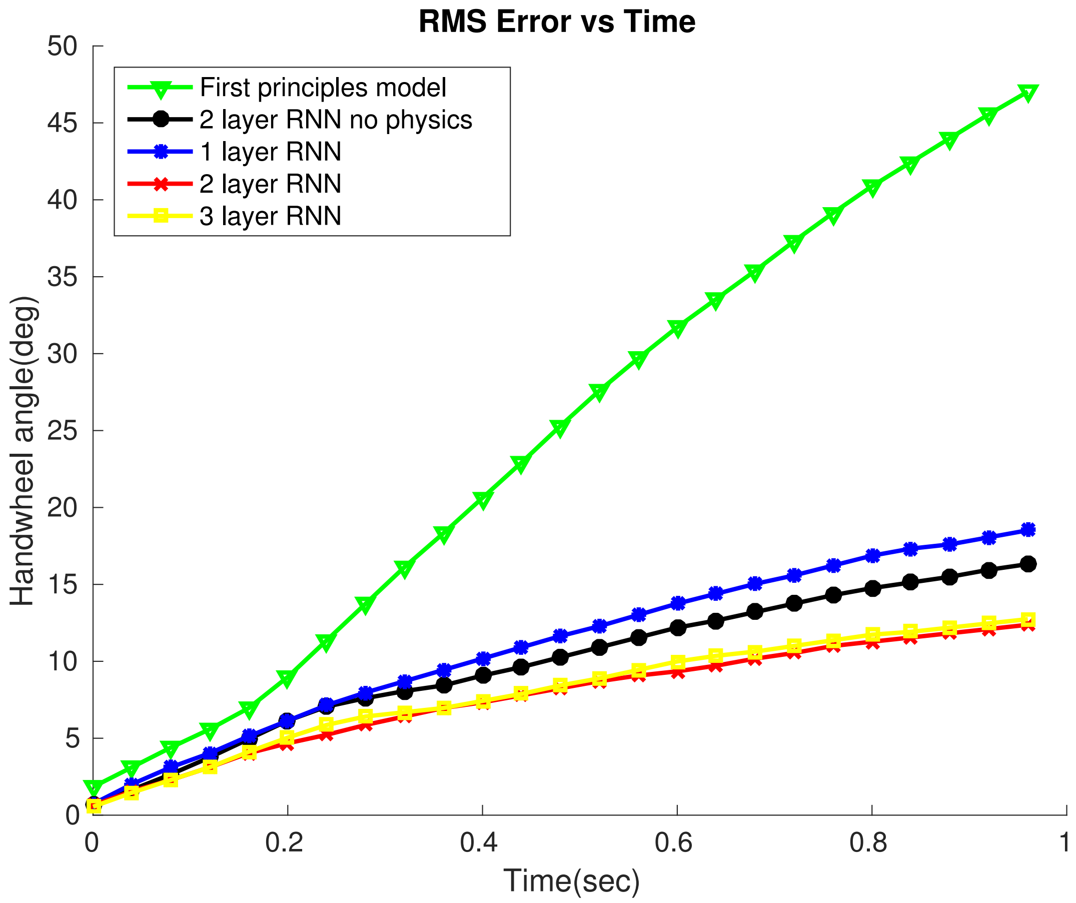


Figure 5.4: Averaged RMS Error between propagated and measured hand wheel angle along the trajectory for all the test segments plotted for different dynamics models.

trajectory comprises of a reference steering angle δ_r and steering rate $\dot{\delta}_r$. The feedforward steering torque estimates the necessary command to track the desired reference and thereby improves tracking performance when compared to applying the PID component alone. The procedure to compute feedforward steering torque using different models is discussed next.

5.1.4.1 Inverting First Principles Model

Here, the feedforward steering torque for controlling the steering system is computed by inverting the first principles model. Denoting the power steering dynamics at a given longitudinal velocity by $P(\tau_s) := g(\tau_s, v_x)\tau_s$, the input torque required to track a steering reference trajectory is computed as

$$\tau_s^* = P^{-1}\left(-k_p\delta_r - k_d\dot{\delta}_r - k_a F_{yf}\right).$$

The desired steering angle, steering rate and forward velocity along the trajectory are assumed to be known during inversion. The tire force F_{yf} is computed based on reference trajectory as

$$F_{yf} = k_f m v_x \dot{\phi},$$

where the gain k_f is the ratio of the distance between the center of mass to front wheels to the wheelbase. The feedforward steering torque for trajectory tracking is chosen to be equal to the computed input torque $\tau_{ff} = \tau_s^*$, since the computed input torque is only dependent on the reference trajectory.

The double derivative of the reference steering angle, the Coulomb friction and the desired steering acceleration are not accounted for when computing the feedforward steering torque to avoid inducing high-frequency oscillations into the steering control.

5.1.4.2 Lookup table

The feedforward steering torque can also be computed using a lookup table under steady state assumptions. The lookup table provides the steering torque required to achieve a steady state steering angle for a given velocity. The data in the lookup table is filled based on experiments in which the steering torque is adjusted to achieve the desired steering angle for a given forward velocity. The feedforward steering torques for a given reference steering angle and forward velocity are then computed by interpolating the data points from the lookup table.

5.1.4.3 NMPC Steering Control

When using the neural network model, the feedforward steering torque is computed by solving an NMPC optimization problem. The optimization problem is set to track a reference steering trajectory given by $\tilde{s}_{0:N}$ using the neural network model. The longitudinal velocity is fixed to be equal to the reference velocity from high-level planner during the optimization process. The optimization problem for computing the feedforward steering torque can be written as

$$\tau^*_{0:N-1} = \underset{\tau_{0:N-1}}{\arg\min} \sum_{i=1}^{N} (s_i - \tilde{s}_i)^T P_i (s_i - \tilde{s}_i) + \tau^T_{i-1} R_i \tau_{i-1},$$

$$\text{s.t} \quad x_{i+1} = f_{ph}(z_i) + f_{nn}(z_i),$$

$$s_i = [\delta_i, \dot{\delta}_i]^T, \tilde{s}_i = [\delta_r, \dot{\delta}_r]^T$$

$$\text{Given } \{v_{x0:N-1}, \tilde{x}_{0:N}\}.$$

The first component of the optimal steering torque from NMPC optimization is sent to the PID controller as feedforward steering torque ($\tau_{ff} = \tau^*_0$). If there are delays in sending the torque command, the feedforward steering command can correspond to the future stamped steering torque as in $\tau_{ff} = \tau^*_{delay}$. The NMPC optimization is performed using a Stage-wise Newton method as explained in chapter 2.

5.1.5 Results

The MPC controller for steering dynamics has been tested on a passenger car equipped with a high-precision GPS system, and an onboard compute stack. The low-level automotive system is accessed through a CAN bus to send engine throttle and steering torque commands. The onboard compute stack consists of two computational modules: high-level computer and low-level microcontroller. The high-level computer performs the high-level planning and provides a steering reference trajectory as an output. It also produces the feedforward steering torque necessary for tracking the steering reference trajectory using NMPC optimization described in section 5.1.4. The low-level microcontroller runs a steering PID controller that outputs a torque command based on the input steering reference and feedforward torque. Communication between the high-level computer and the low-level microcontroller is handled through a separate CAN bus.

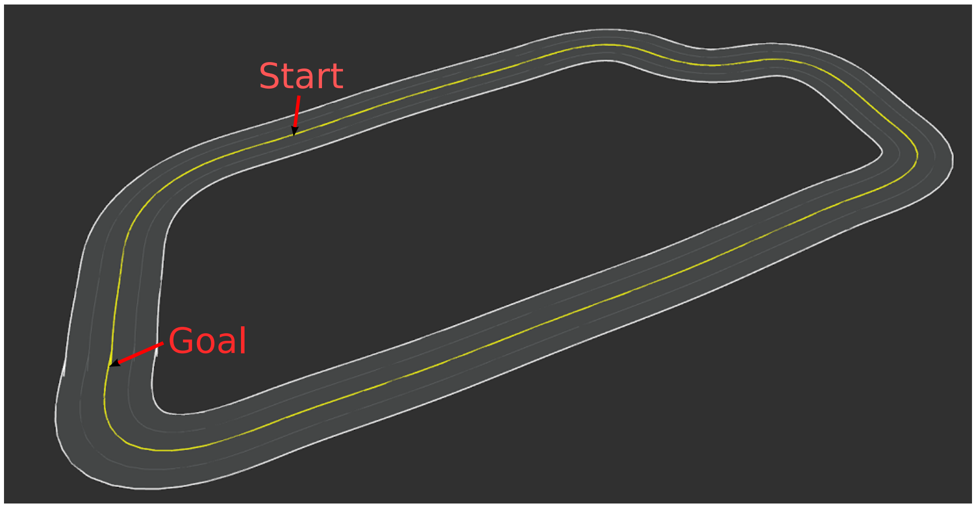


Figure 5.5: Test track used for autonomous driving experiments

The goal of the experiments is to closely track the desired steering reference trajectory(denoted by a series of handwheel angles) in a small test track as shown in Fig 5.5. The steering controller is tested at longitudinal velocities of 5mph and 10mph under a maximum lateral acceleration of 3.5m/ss. Large lateral acceleration and low speeds are used to test the system, as these conditions tended to cause worse steering tracking performance. The effect of adding a feedforward steering torque using different models on the steering tracking performance is shown in Table 5.1. Not including a feedforward in the controller results in the largest RMS error, as expected. Adding the lookup feedforward model improves the performance only slightly, probably because these are more dynamic rather than steady state maneuvers. Computing the feedforward steering torque using the first principles model, which incorporates these dynamic effects, outperforms the lookup table model.

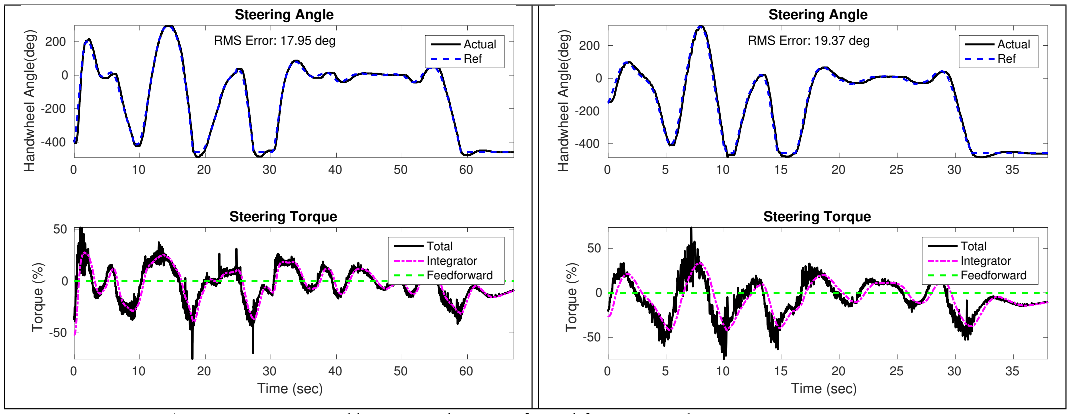
The neural network performs comparably to first principles model with regards to the RMS error. It also performs better than the first principles model at low velocities. The first principles model is not able to model the road-wheel interactions correctly at low velocities. This is likely because at lower velocities, the power steering system provides larger assist, and that effect is difficult to model directly without knowing the underlying power steering software algorithm. The neural network, however, benefits from its own internal representation based simply on training data.

Controller/Longitudinal velocity	**5mph**	**10mph**
No Feedforward	18.63	18.61
Lookup table	16.43	14.88
First principles model	12.48	9.56
Neural network model	10.21	9.53

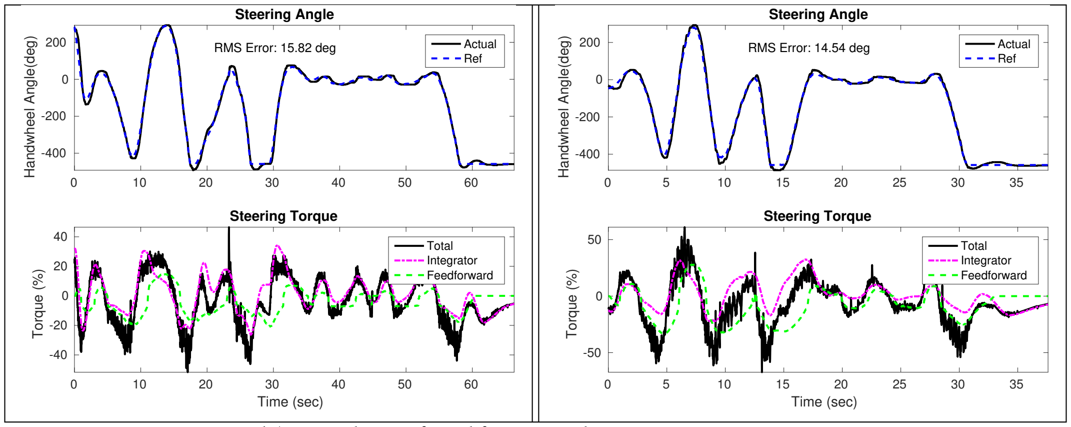
Table 5.1: Experimental RMS Handwheel error (degrees) for different models

The steering performances of different steering controllers are shown in Figures 5.6, 5.7 for a single trial at 5mph and 10mph longitudinal velocities. The initial steering angle differences are matched across different controllers before computing the RMS error to remove the effect of various initial conditions. The PID controller without feedforward steering torque has the highest RMS steering error as seen in Table I, and as a result of the larger errors, the integrator performs much of the tracking task. Adding a lookup table offers marginal improvement, since the maneuvers are highly dynamic. The first principles model and the neural network model outperform the other methods with regards to RMS error. The magnitude of the integrator is also small and most of the tracking performed by the feedforward steering torque.

5mph Trials	10 mph Trials



a) PID controller without feedforward steering torque

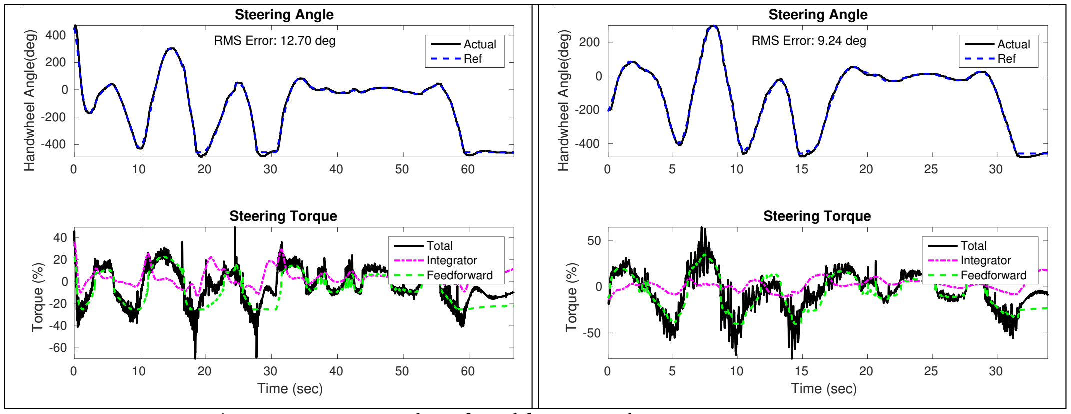


b) Lookup feedforward steering torque

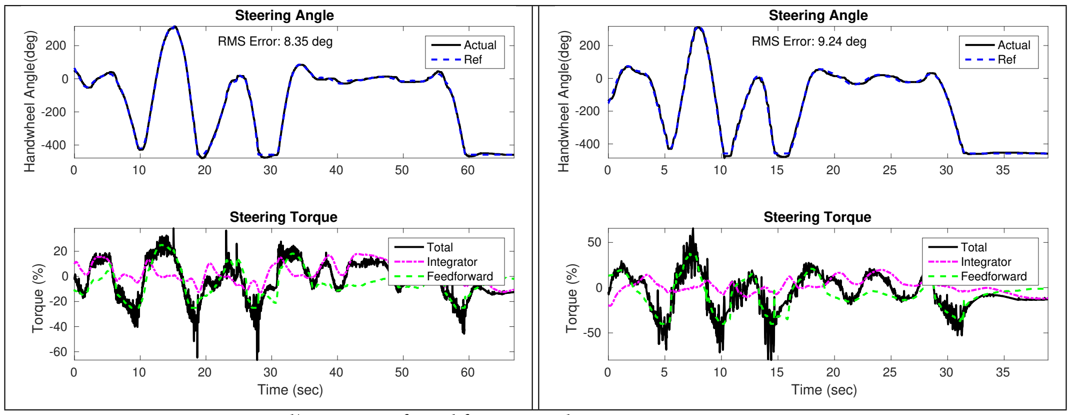
Figure 5.6: Comparison of steering performance using PID and Lookup steering control methods

5.1.6 Conclusions

The use of Recurrent Neural Network model, for modeling and control of a vehicle's steering system has been presented. The creation of the neural network



c) First principles feedforward steering torque



d) RNN feedforward steering torque

Figure 5.7: Comparison of steering performance using Feedforward and Neural network steering control methods

model did not require domain specific knowledge about the power steering module and steering system dynamics. It used only minimal knowledge of the nominal car dynamics to improve the prediction capability. The resulting neural network model outperformed a first principles model in the long-term prediction of steering dynamics and operated equally well in generating a feedforward reference command for control. These results demonstrate that a simple RNN, augmented with simple dynamics information but lacking domain specific knowledge, can be suitable for dynamical modeling and control of a vehicle steering system. The current system is limited to learning RNN model offline and computing feedforward torque online. Future work will include learning the neural network model online and testing the vehicle on changing road conditions.

5.2 Neural Network Modeling for Controlling an Aerial Vehcle

5.2.1 Introduction

This section considers the problem of designing a controller for accurate trajectory following of an Unmanned Aerial Vehicle (UAV) equipped with a multi degree-of-freedom (DOF) arm. We use a Recurrent Neural Network (RNN) to model the dynamics of the system and perform NMPC optimization on the learned model to track reference trajectories.

Aerial manipulation has potential application in package delivery [11], bridges and furnace inspection [81, 92, 84], cooperative transportation [153], stippling [58], pruning tree branches [157], and aerial sampling [167]. Such applications often require precise control of the end-effector position. For example, small errors in the end-effector position can result in failure to pick up a package or retrieve a sample.

The control of an aerial manipulator is complicated primarily due to the interactions between the multi-DOF arm and the UAV platform. Several control techniques have been developed in the past for controlling the aerial manipulator [109, 90, 75]. These controllers usually consider the inputs to the system as the body torques and the body thrust applied by the UAV platform and the joint torques applied by the arm. In practice, off-the-shelf UAVs are controlled using an on-board autopilot such as DJI A3 [47] which controls the orientation of the UAV and a scaled thrust along the body z-axis of the UAV. Similarly, the arm is controlled using a servo motor that regulates the joint angles or joint velocities. The control logic of the autopilot and the servo motors is usually not known to the user for modeling purposes. Even if the logic is known, the exact forces and torques applied to the system are unknown since the controllers only use an approximate actuator model [53].

We employ a Recurrent Neural Network (RNN) to model the complete coupled aerial manipulator dynamics. Using an RNN allows for modeling the unknown control logic and the actuator dynamics based on sensor measurements. The RNN model is trained by piloting the aerial manipulator manually and collecting sensor data along the piloted trajectories. The RNN model is augmented with a feedforward model that accounts for the well-known rigid body kinematics and a simplified second order PD control model. The augmented RNN model is able to predict the aerial manipulator position accurately with an RMS error of 5 centimeters for an open-loop prediction horizon of 1 second based on a typical test dataset. The trained RNN model is then used in a Nonlinear Model Predictive Control (NMPC) framework to control the aerial manipulator. We experimentally compare the performance of the RNN model against a standard feedforward model in tracking a spiral reference trajectory for the UAV and sinusoidal reference trajectory for the arm.

Related Work

Aerial manipulator systems have been studied extensively in the past. Orsag, Korpela, and Oh [170] showed design and control of a UAV attached with a multi-DOF manipulator. Kondak et al. [113] showed the control of an industrial 7-DOF arm attached to a helicopter system. Cooperative transportation of aerial system has also been addressed in [56]. Kondak et al. [116] further showed cable transportation of load using three helicopters simultaneously. Nguyen et al. [160] simplified the cooperative transportation problem by decoupling the rotational dynamics of individual quadrotors using a ball joint attached at it's base. Most methods based on traditional non-adaptive control require careful tuning and in general lack robustness to large changes in the dynamics, for instance due to wind gusts, propeller down-wash, or contacts.

NMPC optimization is a control approach that solves an optimization problem at every step by propagating the system dynamics and minimizing a user defined cost function along the trajectory. It accounts for system constraints and could handle changing system dynamics. NMPC optimization for a higher dimensional system such as an aerial manipulator is computationally expensive and has been initially limited to simulations. More recently, Nikou et al. [163] showed cooperative transportation of a load using multiple quadrotors while avoiding collisions and singularities using simulated dynamics. Neunert et al. [159] and Lunni et al. [136] showed NMPC optimization onboard a multirotor vehicle. The effectiveness of the NMPC methods depend on the accuracy of the model used. There is a non-trivial effort involved in identifying the model required for the NMPC optimization and updating the system parameters as the aerial manipulator interacts with the environment. We consider constructing such complex models using a RNN to predict the system motion.

The application of neural networks to aerial robots has generally been limited to vision-based recognition and path planning applications (e.g. Carrio et al. [30], Maciel-Pearson and Breckon [138], and Heylen et al. [76]). Recently, end-to-end learning models have been used to control robot systems based on raw sensor data (Kelchtermans and Tuytelaars [97]). These methods are difficult to generalize to additional sensing modalities and to handling changes in the system dynamics. In this work, we use a RNN model to learn the dynamics of the aerial manipulator and use NMPC optimization for controlling the robot based on the predicted dynamics. This two-stage approach allows us to reliably verify the predictive capability of the RNN before using the controller thereby reducing the chance for potentially unsafe robot behavior.

Deep NMPC which combines a deep RNN with NMPC optimization has been proposed by Lenz, Knepper, and Saxena [127] for a robot cutting task. Unlike the RNN proposed in that work, we employ a much simpler model using a 9-DOF IMU, joint angles obtained from a potentionmeter, commands applied to the robot, and also combining the *unknown* learned dynamics with a *known* feedforward model for which we also learn the gains. Further, this model uses a known dynamic state which can be estimated using traditional estimators. It also allows for substitution of sensors without the need to retrain the network.

Other related work includes the use of an RNN model in an NMPC approach to control the steering dynamics of a passenger vehicle in [60]. The proposed RNN model predicts the error between predicted feedforward state and the measured state. In contrast, the RNN architecture used here produces accelerations as output which is integrated through a separate integration approach. This makes the output trajectories smoother and thereby the controls produced are also smoother.

Contributions

We propose a novel RNN architecture that computes the acceleration of a robotic system and integrates the accelerations to find the robot state at the next time instant. The integration of the accelerations is performed in a separate integration stage that takes as inputs the robot state, control and time-step of integration. We use a semi-implicit integration approach to maximize accuracy in prediction. The RNN architecture is augmented with a feedforward model to increase the predictive capacity of the model considerably with only a small increase in the number of parameters. A smaller-size model is also critical for fast real-time control optimization.

The resulting RNN is employed as a predictive model in an NMPC framework to track reference trajectories. A second-order stage-wise Newton [19] trajectory optimization method is employed which has a $O(N)$ complexity where N is the number of trajectory time-steps. The resulting algorithm can operate at 100 Hz using a 2 layer RNN model with 16 and 8 nodes running on an embedded i5 computer onboard the UAV platform. We then perform empirical evaluation of the controller on a UAV platform tracking spiral reference trajectories. The NMPC approach achieved an approximate position accuracy of 0.1 meters and joint angle accuracy of 0.1 radians when tracking spiral reference trajectories for the UAV and sinusoidal trajectories for the joint arm simultaneously.

5.2.2 Modeling dynamics

We employ a quadrotor UAV platform equipped with 2-DOF manipulator arm as shown in Fig. 5.9. The quadrotor platform is an underactuated system consisting four co-axially aligned propellers. These propellers can control independently the three body torques around the principal axes of the body and thrust along the body z-axis. We assume that the quadrotor's orientation given by Euler angles ($\xi \in \mathbb{R}^3$) and the thrust $f_z \in \mathbb{R}$ are controlled using an autopilot module and the joint angles of the arm ($r \in \mathbb{R}^2$) are controlled using servo motors attached to the two joints.

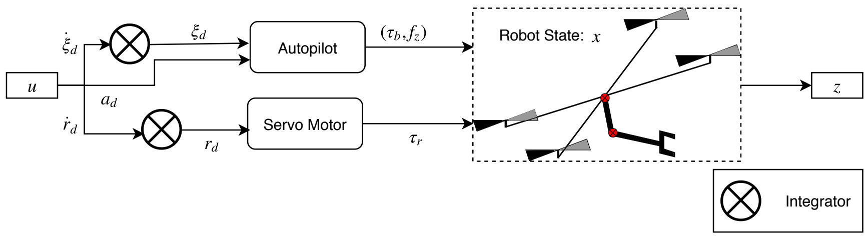


Figure 5.8: Schematic of the aerial manipulation system with inputs u, outputs z, and robot state x.

We assume the autopilot dynamics is a second order system i.e. the autopilot applies body torques ($\tau_b \in \mathbb{R}^3$) based on the commanded Euler angle rates ($\xi_d \in \mathbb{R}^3$) and the desired Euler angle rates ($\dot{\xi}_d \in \mathbb{R}^3$). It is also assumed that the thrust f_z applied by the autopilot is directly related to commanded thrust $a_d \in \mathbb{R}$. Hence, we assume the inputs to the autopilot system are $(\dot{\xi}_d, a_d)$ and treat ξ_d as part of the state. Similarly, assuming the servo model is a second order system applying joint torques $\tau_r \in \mathbb{R}^3$, the inputs are chosen to be desired joint angle rates $\dot{r}_d \in \mathbb{R}^2$ and the desired joint angles $r_d \in \mathbb{R}^3$ are assumed to be part of the state.

The quadrotor platform is attached with motion capture markers that are tracked using motion capture cameras which provide the position $p \in \mathbb{R}^3$ and the orientation ξ in an inertial frame. The servo motors provide feedback in terms of joint angles $r \in \mathbb{R}^2$. The schematic of the aerial manipulator shown in Fig. 5.9 shows the inputs and outputs associated with the aerial manipulation model. The combined sensor measurements $z \in \mathbb{R}^8$ and the controls $u \in \mathbb{R}^6$ are shown in (5.1).

$$z = [p, \xi, r]^T, \quad u = [a_d, \dot{\xi}_d, \dot{r}_d]^T \tag{5.1}$$

5.2.2.1 Network Architecture

The goal of the RNN is to predict the outputs at the next step z_{i+1} (i.e. the sensor measurements) given the control at the current step u_i and the state x_i where i denotes the sequence index which in our context denotes the time elapsed. We want to use as much prior information about the model as possible to minimize the amount of training data required and reduce the size of networks. Smaller networks enable lower computational effort during NMPC optimization. To use prior information known about the model, the state x is chosen manually to denote all possible feature that are expected to provide z_{i+1}. For the aerial manipulation system, the state x is selected to denote the full dynamical state of the system, i.e. the position of the quadrotor p, the velocity of the platform $v \in \mathbb{R}^3$, the Euler angles ξ, the rate of Euler angles $\dot{\xi}$, the commanded Euler angles ξ_d, the joint angle r, the joint velocities $\dot{r}$, commanded joint angles r_d and a scaling coefficient on the commanded thrust denoted as $k_t \in \mathbb{R}$. The scaling coefficient is a part of the feedforward model that predicts the body z-axis acceleration given the commanded thrust as shown in (5.3). The overall state is given as

$$x = [p, \xi, v, \dot{\xi}, \xi_d, r, \dot{r}, r_d, k_t] \tag{5.2}$$

The mapping between the state and the sensor measurements is denoted by $g(\cdot)$ and simply selects the correct sensor channels from the state, i.e. $z = g(x) = [p, \xi, r]$. Since the state is manually selected, its propagation is non-trivial and cannot be accomplished using a small number of fully connected layers. We can rely on prior information to propagate the state and rely on fully connected layers to only learn the residual dynamics. The prior information is encoded into the architecture in two phases: "Force prediction" and "Integration" as shown in Fig. 5.8. In the first phase, the state and the control are used to generate the resulting accelerations produced on the system. These accelerations are corrected using fully connected layers to account for unmodeled dynamics. The corrected accelerations are then integrated using a semi-implicit integration approach. The integration phase of the model only depends on the time step of integration and usually does not have parameters that need to be trained.

Using the network to predict the perturbations in forces and torques makes the system trajectories smooth up to second order which is necessary for smoother control input to the system. Further, using integration which is decoupled from the acceleration prediction allows the same trained network to be used with different time steps during integration. In addition, the integration phase of the model can also incorporate limits on accelerations and velocities that are known before hand. For example, the joint velocities of the arm are bounded by 0.7 radians per second by the servo controller. This can be easily incorporated into the integration phase and does not require learning the same.

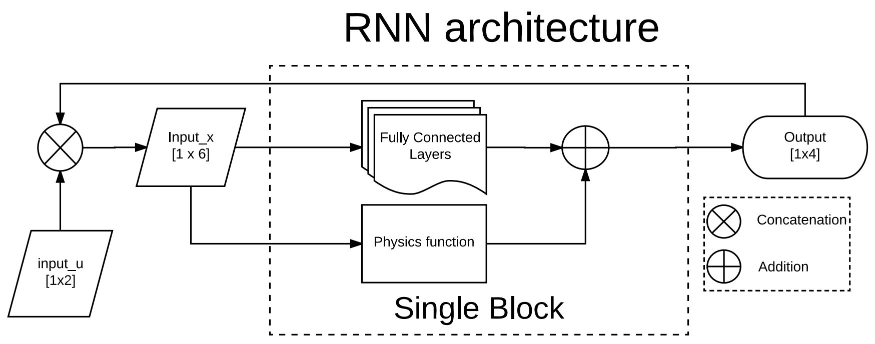


Figure 5.9: Schematic of the neural network architecture

5.2.2.2 Force prediction model

The prior information about the quadrotor system and servo control is used to formulate a model that predicts inertia normalized forces, i.e. accelerations, that are required to propagate current state and control. For producing the angular accelerations on the quadrotor and joint accelerations on the arm, a second order PD control loop is applied between both the commanded and observed Euler angles ξ and the commanded and observed joint angles

r. The acceleration on the quadrotor platform is obtained by scaling the commanded thrust and compensating for gravity ($g = [0, 0, 9.81]^T$). The overall feedforward dynamics can be written as

$$a = k_t a_d \bar{z} - g, \tag{5.3}$$

$$\ddot{\xi} = -k_{p_\xi}(\xi - \xi_d) - k_{d_\xi}(\dot{\xi} - \dot{\xi}_d), \tag{5.4}$$

$$\ddot{r} = -k_{p_r}(r - r_d) - k_{d_r}(\dot{r} - \dot{r}_d), \tag{5.5}$$

where $\bar{z}$ denotes the body z axis of the quadrotor in inertial frame which is obtained from the Euler angle ξ. The concatenated acceleration vector is denoted as $\tau = [a, \ddot{\xi}, \ddot{r}] \in \mathbb{R}^6$. The unknown parameters in the model are the proportional and derivative gains for Euler angles ($k_{p_\xi} \in \mathbb{R}^3, k_{d_\xi} \in \mathbb{R}^3$) and joint angles ($k_{p_r} \in \mathbb{R}^2, k_{d_r} \in \mathbb{R}^2$). These parameters are optimized along with the fully connected layer weights and biases. The force prediction model does not include the interactions between the arm and UAV since such a model depends on moment of inertia of the platform which is not observable without having access to the joint torques and body torques applied.

5.2.2.3 Residual dynamics

The accelerations produced by the feedforward model are not accurate since it incorporates only a simplified model of the actual dynamics and neglects the interactions between the quadrotor and the arm. The accuracy of the feedforward model can be improved by adding fully connected layers that predict the difference between the actual accelerations and the accelerations generated by the feedforward model, denoted by $\delta\tau \in \mathbb{R}^6$. The input to the network is given by the feedforward accelerations τ, the current state x, and controls u as shown in Fig. 5.8. It is necessary for us to scale the inputs before passing them into the fully connected layers to avoid saturating the neural network activation functions. We use batch normalization to automatically figure out the scale of the inputs. It is also safe to assume that the horizontal position of the quadrotor does not affect the acceleration of the quadrotor. Hence the horizontal position is neglected before passing the state into the fully connected layers. We use dropout layers on the intermediate fully connected layers and the residual correction $\delta\tau$ to ensure the feedforward dynamics is also trained even if the network produces noisy corrections.

5.2.2.4 Integration

The integration block integrates the corrected accelerations $\bar{\tau} \;= \tau + \delta\tau$ for a specified time step to obtain the state of the robot at next step. We used a semi-implicit integration approach for integrating the accelerations. Under this approach, the velocities at the next step are found by integrating the accelerations, and the averages of the current and predicted velocities are used to integrate the positions forward. A similar approach is used for integrating joint angles as shown below:

$$\dot{r}_{i+1} = \dot{r}_i + \delta t_i \ddot{r}_i, \tag{5.6}$$

$$\dot{r}_{i+1} = \min(\dot{r}_{i+1}, \dot{r}_{max}), \tag{5.7}$$

$$r_{i+1} = r_i + \delta t_i \frac{1}{2}\left(\dot{r}_{i+1} + \dot{r}_i\right), \tag{5.8}$$

where δt_i denotes the time step of integration and $\dot{r}_{max} \in \mathbb{R}^2$ denotes the maximum joint velocity. The position p_{i+1} and orientation ξ_{i+1} of the quadrotor are integrated in a similar manner. Note that the Euler angles ξ wrap around 2π radians which is incorporated into the integration approach, as well.

The thrust scaling gain k_t is assumed to be constant during integration, i.e. $k_{ti+1} = k_{ti}$. The thrust gain usually changes with the mass of the vehicle and the battery voltage of the vehicle. These effects are not observable from the predicted measurements alone. Learning to predict the change in thrust gain did not provide any improvement in the results during training.

5.2.2.5 Estimating System State

So far in the network architecture, we predicted the measurements at the next time step given the controls u_i and the state x_i at the current step. Thus, in order to predict sensor measurements for a sequence of time steps $t_{0:N}$, we require the controls along the time steps $u_{0:N-1}$ and the state of the system at the first step x_0.

Since we manually selected the dynamic state of the system, we can use traditional estimation methods to find the state given the sensor measurements. The position, orientation and joint angles of the state are obtained from the sensor measurements directly. The velocities, Euler angle rates, and joint velocities are obtained by filtering the finite difference differentiated sensor measurements. The commanded Euler angles and joint angles are obtained by integrating the commanded rates and assuming the system starts with commanded angles equal to the measured angles. Finally, the thrust gain

k_t is obtained by filtering individual measurements of thrust gain. A single measurement of thrust gain is obtained by inverting (5.3) i.e $k_t = \bar{z}^T(a + \mathbf{g})/a_d$. The accelerations in the equation are obtained by finite differentiating measured velocities and smoothing them with an exponential filter.

5.2.2.6 Training

During the training phase, both the feedforward model parameters $\theta = [k_{p_\xi}, k_{d_\xi}, k_{p_r}, k_{d_r}]$ and the weights associated with the fully connected layers are learned together. Since both the fully connected layers and the feedforward model are trying to generate the same quantity, i.e. accelerations for the dynamic system, there will be multiple solutions to the parameters which can produce the same accelerations. Thus, we incorporate a prior on the feedforward model parameters to ensure the model is generalizable. This prior has been obtained by training the model using only the feedforward dynamics without adding the residual dynamics. A dropout layer has also been incorporated on the output of the fully connected layers to ensure the feedforward model is applying the right accelerations even when the residual network output is noisy.

The training data consists of several sequences of human piloted quadrotor data along with sinusoidally oscillated arm data with a variety of frequencies, phases, and offsets. The data has been verified to cover as much of the state space as possible keeping in mind the safety of the vehicle. These sequences are further split into several smaller segments with a fixed horizon length of 1 second with a back propagation length of N segments (In our case we used $N = 50$ segments since the motion capture data is sampled at 50 Hz). The state of the system is estimated along the entire sequence of collected trajectories and the starting estimated state for each sequence has been stored for training purposes. Finally, stochastic gradient descent has been applied on the cost function shown in (5.9) where the predicted sensor measurements are obtained by unrolling the neural network using the applied controls and initial sensor measurement for the sequence. The cost function consists of the error between predicted sensor measurements $z_{1:N}$ and the observed sensor measurements $\bar{z}_{1:N}$ in addition to regularization costs for the feedforward parameters θ.

$$L = \sum_{i=0}^{Nb} \frac{1}{N} \sum_{j=q}^{N} \left(z_j - \bar{z}_j\right)^T \Sigma_z^{-1} \left(z_j - \bar{z}_j\right) + \frac{1}{2}(\theta - \theta_p)^T \Sigma_\theta^{-1} (\theta - \theta_p), \tag{5.9}$$

where Σ_z is the covariance in sensor measurements, θ_p is the prior on the feedforward model parameters, Σ_θ is the covariance associated with the prior.

5.2.2.7 Prediction Results

The model obtained from training has been verified on a test data set. Fig. 5.10 shows the position RMS errors against the length of the trajectory (arc length). The feedforward model predicts smaller length trajectories better and its performance degrades with the length of the trajectory. This behavior is also exhibited by neural networks without including the feedforward network. The performance of the RNN augmented by feedforward model does not degrade with arc length and is almost constant.

The RMS error along each of the sensor channels is shown in Fig. 5.10. It

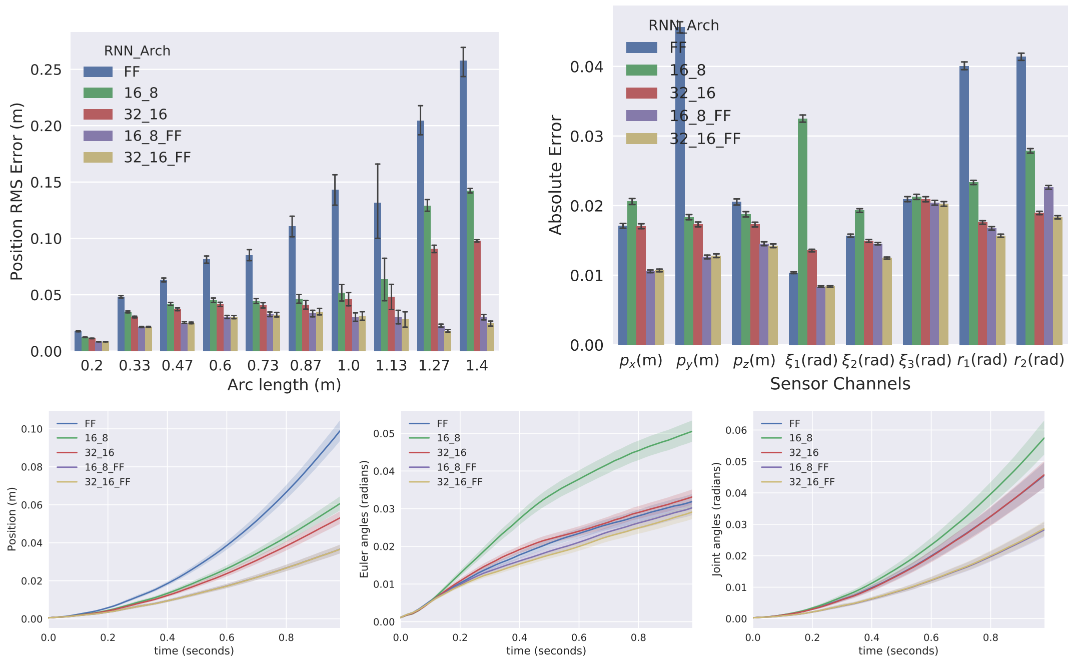


Figure 5.10: RMS errors along with 95 percent confidence interval on test dataset. (a) Position RMS error against length of the trajectory. Trajectories that have larger arc length are harder to model. (b) RMS Error for each sensor channel. (c) Position RMS error against time (d) Orientation RMS error against time (e) Joint angle RMS Error against time

can be observed that without a feedforward model, the smaller RNN network shown in green has a large Euler angle error compared to a pure feedforward model. Using a larger RNN network overcomes this issue but requires more computational effort which increases the optimization time during NMPC. The RNN model augmented with feedforward dynamics improves only slightly when the number of nodes in the network are doubled. This suggests that using the smaller RNN network combined with a feedforward model is sufficient for NMPC control.

The RMS errors of the sensor channels against the time horizon are shown in Fig. 5.10. The position errors increase nonlinearly with time since the position of the UAV is predicted based on accurate prediction of other states such as velocities and accelerations. The Euler angle prediction error increases linearly for most of the RNN and feedforward models. The joint angles on the other hand saturate to an RMS value that decreases with an increase in RNN prediction capacity. The RMS errors for all the sensor channels start out with a very small value and degrade gradually. This is a necessary property for the model to be useful in an NMPC approach.

5.2.3 Experiment Results

We employed a Stage-wise Newton method [19] to solve the trajectory optimization problem.

The NMPC optimization has been tested on a DJI Matrice UAV equipped with a 2-DOF arm to track spiral reference trajectories for the UAV and sinusoidal reference trajectories for the arm.The spiral reference has been chosen to have a radius of 0.2 meters in x and y directions and has a pitch of 1.0 meters and a frequency of 0.2 Hz. The sinusoidal joint angle reference trajectories are chosen to have an amplitude of 0.2 radians with a frequency of 0.2 Hz and an offset of -1 radians and 0.3 radians for the two joints respectively. The NMPC optimization runs at a frequency of 50 Hz onboard a core-i5 NUC mini-PC. The UAV pose is obtained from motion capture cameras and joint angles are obtained from servo feedback.

Figure 5.11 shows the position tracking error for a sample trajectory using MPC with the feedforward model and RNN model. Figure 5.12 shows the joint angle tracking error for the same sample trajectory. The mean absolute

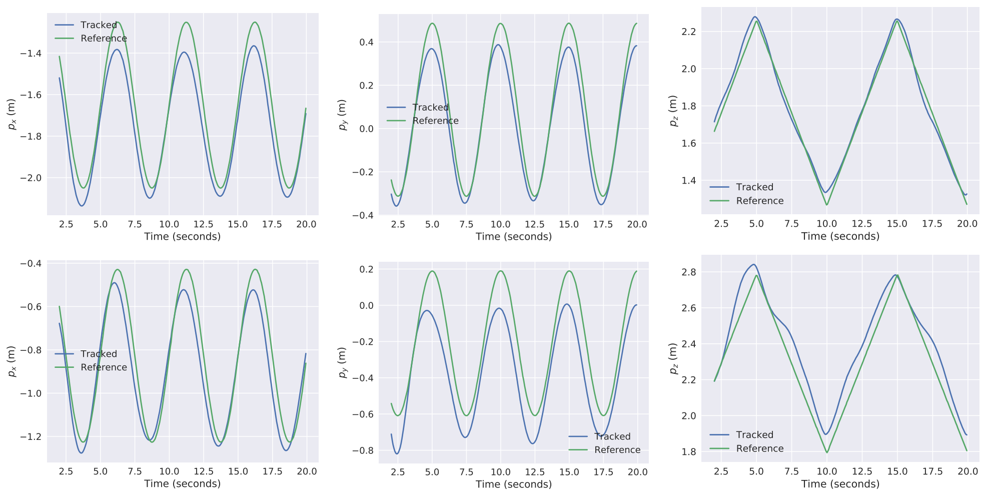


Figure 5.11: Top row shows tracking error for MPC using feedforward model and bottom row shows the error using RNN model

errors for each position axis are shown in Fig. 5.13. As can be observed, the RNN and feedforward model perform equally well in tracking the UAV trajectory. The RNN model slightly underperforms in tracking joint angles. A possible reason for the underperformance is that the feedforward model is very good at predicting the joint angle trajectories, and the addition of a neural network is adding noise to the model optimization. Another possible concern is that the controls being optimized over are not constrained to be from the same distribution as the training data sets. Hence the NMPC optimization can find controls that minimize the cost function but the controls might not be physically meaningful. In-spite of these issues, the RNN model performed well in tracking spiral reference trajectories with an approximate position accuracy of 0.1 meters as shown in Fig. 5.11 and joint angle accuracy of 0.1 radians as shown in Fig. 5.12.

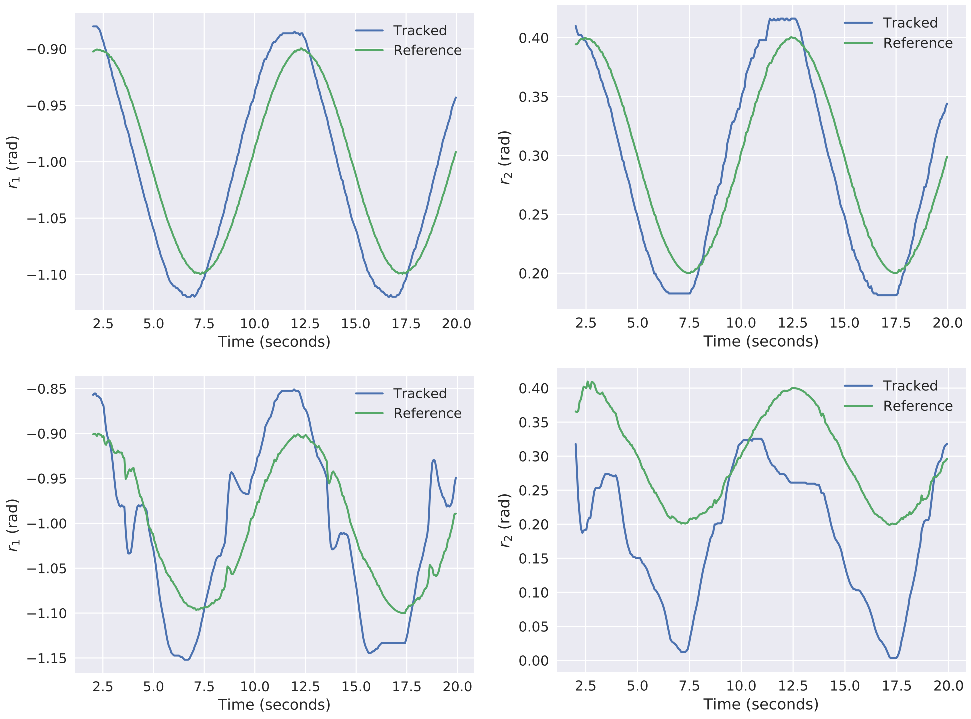
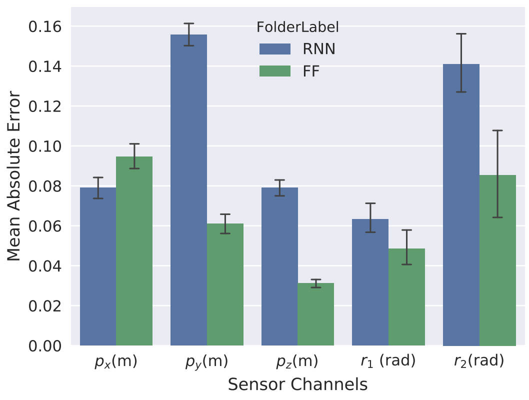


Figure 5.12: Top row shows the joint angle errors for MPC using feedforward model and bottom row shows the error using RNN model



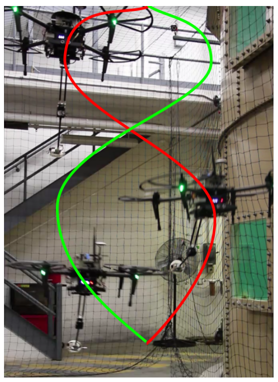


Figure 5.13: Bar plot shows the absolute error along different axes along with 95% confidence intervals. The picture shows the spiral trajectory taken by the quadrotor overlayed on the quadrotor path

5.2.4 Conclusions

In conclusion, we demonstrated the use of an RNN model to predict the coupled dynamics of an aerial manipulator. Combining the RNN model with a feedforward model encoding the known dynamics of the system improved the prediction performance with a small increase in the number of parameters. We employed the RNN model and the feedforward model in an NMPC framework to track reference trajectories for the arm and the UAV. The RNN model performed comparable to the feedforward model and did not provide any notable improvements in tracking performance. Future work should learn to model physical interactions with the environment and applying the NMPC framework to handle external disturbances safely.

Chapter 6

Autonomy Software Framework

6.1 Introduction

In this chapter, we introduce a software framework for combining different low-level controllers such as the robust NMPC controllers developed so far to perform complex tasks in a safe way. We focus on applying the software framework to demonstrate package sorting using aerial vehicles. Vertical take-off and landing (VTOL) vehicles such as quadrotors have gained a lot of attention recently due to their agility and ability to navigated in remote and cluttered environments. Current research suggests that VTOL vehicles attached with manipulators, known as aerial manipulators, are attractive for numerous applications, including package transportation [11], collaborative load transportation [153], collaborative construction, and structural maintenance applications [2, 1]. Many of these applications require interactions between multiple software components and hardware subsystems while navigating cluttered environments to achieve a desired goal. There are also performance constraints on tasks which would require navigating through the environments quickly. In such a scenario, the safety and reliability of the overall system under software and hardware failures is critical. In particular, the task of aerial manipulation is non-trivial since it involves underactuated quadrotor systems combined with multi-degree of freedom manipulators interacting with the environment. We propose a two-fold approach: on the software side, a fault tolerant state machine framework that implements several controllers for aerial manipulation and on the hardware side, a novel magnetic gripper that tolerates end-effector error up to 2 cm while grasping. The result is a reliable aerial manipulation system with a high probability of picking objects (90%) and tolerance to a wide variety of errors.

6.1.1 Related Work

Past research has focused on developing control algorithms and manipulators specifically for aerial manipulation (e.g. Bellicoso et al. [17], Suarez, Heredia, and Ollero [216], and Pounds, Bersak, and Dollar [179]). While results have been reported separately for various aspects of aerial manipulation such as control algorithms (e.g. Mebarki and Lippiello [149], Mellinger et al. [151], Kondak et al. [115], Korpela et al. [118], and Kim, Choi, and Kim [101]), motion planning, and visual servoing (e.g. Kondak et al. [114], Lippiello et al. [133], and Huang et al. [80]), very few fully-integrated systems that allow the combination of these basic behaviors into complex tasks with fault-recovery have been reported. Current commercially available aerial autonomy suites [48, 150] are limited to basic navigation and observation tasks and not directly applicable to aerial manipulation.

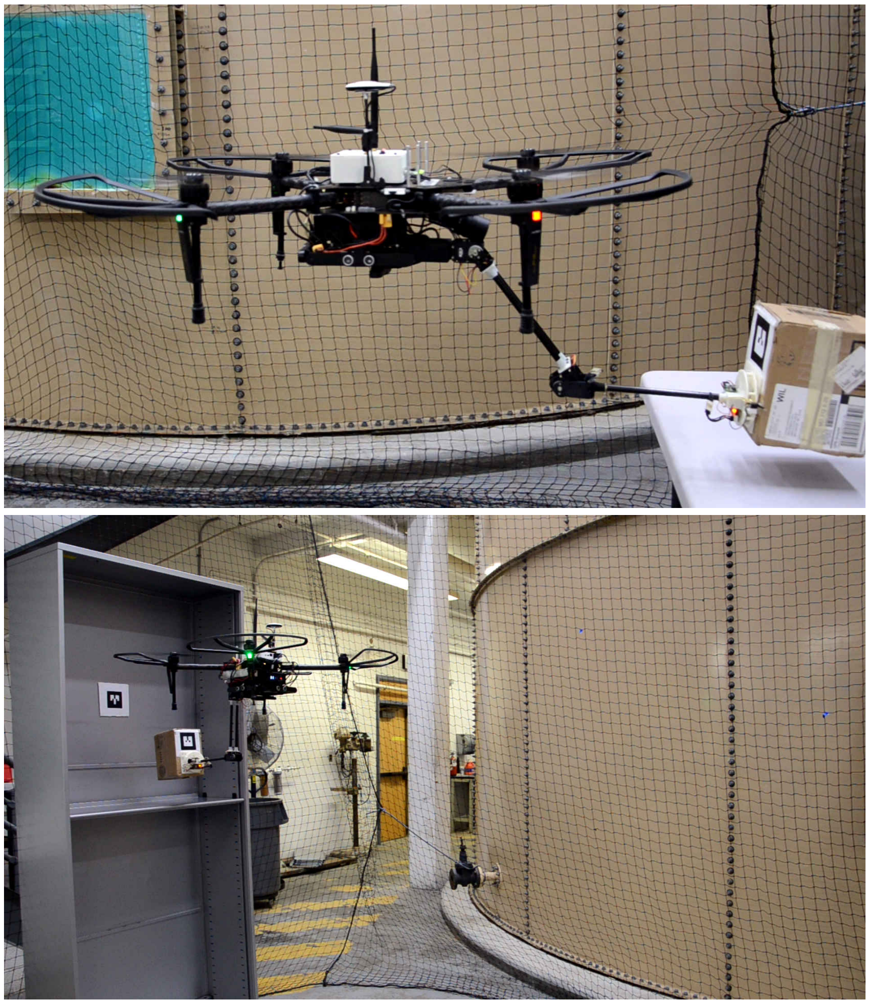


Figure 6.1: Proposed aerial manipulation system picking (top) and placing (bottom) a package.

A few fully integrated applications for aerial manipulation have been introduced in recent years. An aerial manipulation system for moving metallic discs and sheets is proposed by Gawel et al. [61] and Nieuwenhuisen et al. [162]. The system developed by Gawel et al. [61] used an electro-permanent gripper that can turn on and off the magnetic effect by reversing an electric current. In contrast, our work proposes a permanent magnetic gripper solution that can turn on and off by changing the polarity of the magnets using a mechanical servo. This type of gripper does not use energy to hold the object and only requires momentary energy to release objects. Lee et al. proposed a collaborative framework for moving an unknown object in an unknown obstacle ridden environment [124].Kim and Oh [100] developed an aerial manipulation system for lab automation using a parallel manipulator. Orsag et al. suggested a benchmark for different aerial grasping applications [171]. Our work performs two similar benchmark applications: grasping objects from a table and placing them in slots on a shelf.

This chapter proposes a reliable aerial manipulation system, at the core of which lies an autonomy software framework that is robust to controller and hardware failures. We apply the state machine framework to a package sorting application that combines an off-the-shelf quadrotor with a custom built light-weight 2-DOF arm and a magnetic gripper that is tolerant to position error. We implement and compare two different control strategies for picking objects: a PID controller that assumes tight inner-loop attitude control and a Model Predictive Controller (MPC). A novel magnetic gripper is developed that can grasp objects with a tolerance of 2 cm in end-effector position. We tested the entire system through a set of 101 experiments and documented different failure modes that can occur. The software framework is robust enough to complete 85 out of the 101 pick-place trials conducted. Finally, we provide the state machine framework and aerial manipulation controllers as open-source software [212].

6.2 Software Framework

At the core of this system lies a software framework that allows users to easily create autonomy applications by combining modular state behaviors, controllers, and hardware capabilities into domain-specific state machines. The software framework has been designed to:

- combine modular behaviors, such as waypoint navigation and visual servoing, into complex state machines to perform complicated tasks, like object pick-and-place;
- enable robustness to sensor, controller, and hardware failure, through introspection and fail-safe actions;
- provide control methods that adapt to environment changes;
- provide automated tests for controllers and logic systems, independent of their hardware implementation;
- serve as an open-source system for developing complex aerial autonomy applications.

It tightly integrates high-level control strategies for both quadrotors and manipulators with an existing finite state machine library to provide robustness to controller and hardware failures during the task. The software framework consists of two major components– the state machine and the robot system, which are described next.

6.2.1 From behaviors to automatically generated state machine

The state machine and its set of behaviors form the core of our software framework. The state machine logic is specified through a state transition table which consists of a list of tuples that specify the start state, transition event, ending state, transition action, and transition guard.

The states in a state machine denote the different stages during the execution of a task. In the context of aerial manipulation, the states denote different stages of the pick and place task. For example, the “Reaching Goal" state refers to when the quadrotor is navigating to a goal location. Similarly, the “Taking Off" state denotes the quadrotor in the process of taking off from the ground.

State transitions are triggered by an event. For example, a transition from the “Landed" state to the “Taking Off" state is triggered by a "Take-off" event. Events are typically generated by the state machine itself or by users through a graphical interface.

When a state transition is triggered, a guard function first verifies the feasibility of a transition between two states. That is, a state transition occurs only when the guard allows it. If the guard blocks a state transition, the current state will remain unchanged. As an example, in our pick-and-place task, a guard function checks that the output of the object tracking module is valid before transitioning into a “Visual Servoing" state, which attempts to align the quadrotor with an object using visual features.

When the guard allows a state transition to occur, an associated transition action executes. Typically, these actions switch the active controllers, send direct commands to the hardware driver, or configure some aspect of the robot for the new state. The transition actions can be triggered by the user or through state machine logic. These actions can be chained to create more complicated actions and thereby reduce code duplication.

While in a particular state, the system repeatedly executes an internal behavior associated with that state. These behaviors, called “internal actions", trigger specific events on the state machine based on the current robot state. These actions typically perform health checks on the hardware and controllers and check for convergence of active controllers. In the case of the “Reaching Goal" state referred to in transition table 6.1, the internal action checks for the battery status of the quadrotor and triggers an “Abort" event if the battery is low and checks if the quadrotor has converged to the goal location, triggering a “Completed" event if it has converged. Similar to transition actions, the internal actions can also be chained together. Encoding the state machine logic in the internal actions allows for decentralization of state machine logic and reduces code duplication among different state machines.

The state machine has been implemented using the Boost meta state machine library in C++ [22]. The states, actions, and guards in the table are C++ classes that can be reused in different transition tables to form diverse state machine behaviors without code duplication.

A simple example

As an illustration, consider the simple transition table shown in Tables 6.1 and 6.2. A graphical illustration of state transition table is presented in Fig. 6.2. The transition table enables the quadrotor to takeoff, land, and navigate to different waypoints. The user can trigger different events to move the robot around. The transition table safe-guards the robot from erratic behavior. For example, if the robot is on the ground (“Landed"), the user cannot send a waypoint (trigger a “Position Goal" event). Similarly, if the robot battery is low, the “Take-off" event will not transition from the “Landed" state to the “Taking Off" state. The state machine logic is executed through internal actions. These actions trigger different events based on the robot state. For example, if the battery is low enough while hovering, the internal action triggers a "Land" event. Similarly, if the robot has reached the goal position, a "Completed" event is triggered, transitioning the state machine back to the “Hovering" state.

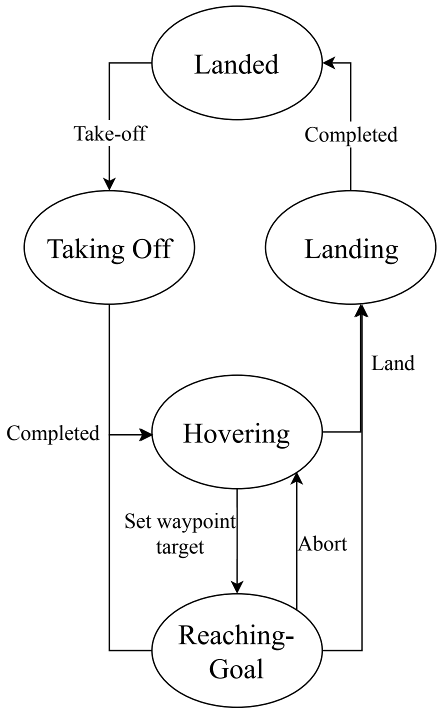


Figure 6.2: Simple state machine state transition diagram

Event	**Source State**	**Target State**	**Action**	**Guard**
Take-off	Landed	Taking Off	Take-off	Check battery level
Completed	Taking Off	Hovering	None	Check altitude
Position Goal	Hovering	Reaching-Goal	Set position goal	Check battery and valid goal
Completed	Reaching-Goal	Hovering	None	None
Abort	Reaching-Goal	Hovering	Abort controller	None
Land	Hovering	Landing	Abort active controller and Land	None

Table 6.1: Simple state machine transition table

State	**Internal Action**
Landed	None
Taking Off	Trigger Completed event when above a certain height
Hovering	Check battery status
Reaching Goal	If battery or controller status is critical, trigger Abort If controller converged, trigger Completed event

Table 6.2: Internal action table associated with a simplified state machine

6.2.2 Robot system

The robot system provides a set of modular capabilities that the state machine executes to drive the robot to a desired state. For example, the pick-and-place task requires several capabilities, such as visual servoing, waypoint following, and manipulator control. Each capability is encoded through software components which interact between each other and with the state machine as illustrated in Figure 6.3. The software components are explained below:

Controllers: The controllers are functions that take in the sensor data and provide controls necessary to drive a system to a desired goal. The controller logic is agnostic of the specific hardware drivers used, allowing for easy testing using simulated dynamics.

Hardware Drivers: At the lowest level, the system relies on a generic interface to hardware drivers. It expects that a particular type of hardware has a general set of capabilities. For example, it expects that all quadrotors can accept roll-pitch-yaw-thrust controls or waypoint commands. The framework then interacts with the hardware without knowledge of the implementation of the underlying driver. This allows users to plug their hardware into existing autonomy applications and controllers by simply conforming their driver to the expected interface. For instance, the software supports a driver for DJI drones (like Matrice and the A3 flight controller) and a generic ROS driver for manipulators.

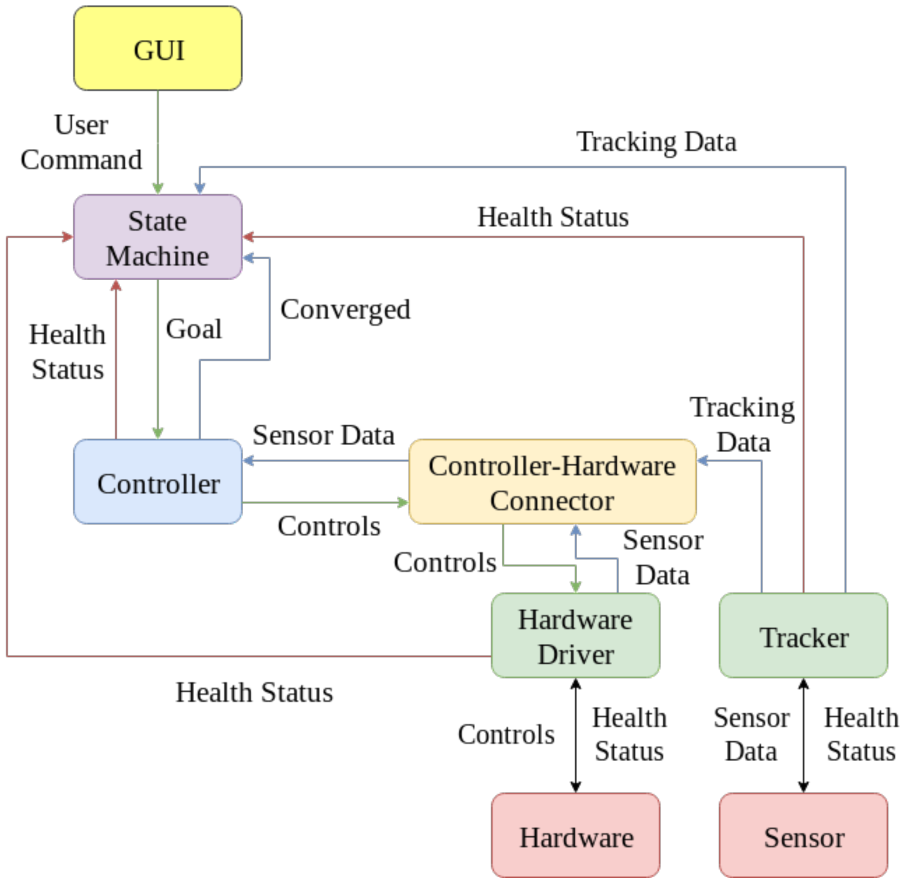


Figure 6.3: Illustration of the interaction between the various software components of the developed framework. The Graphical User Interface (GUI) displays feedback from the individual components, but those connections are left out of the diagram for readability.

Controller-Hardware Connectors: The connector components provide the necessary inputs to the controllers and send the output of the controllers to the hardware drivers. In some cases, the connectors simply apply frame transformations and convert sensor data directly obtained from the hardware to the type expected by the controller. For other cases, such as visual servoing, the connectors obtain the inputs to controllers from other components such as object trackers or state estimators. The connectors then send the controller output to the hardware driver interface. Finally, the connectors also provide a health status that can be consumed by the state machine to trigger different transitions.

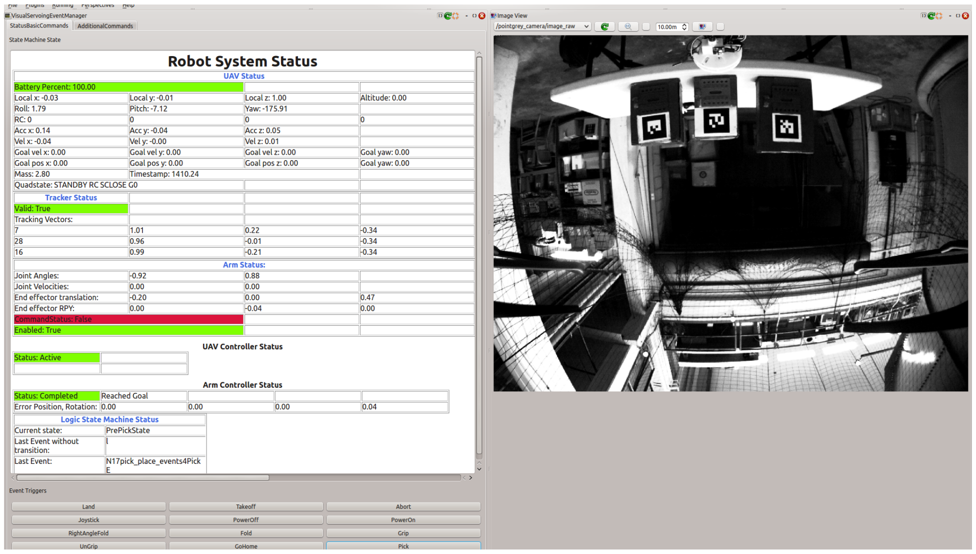
Trackers: Trackers process image data from sensors such as cameras or depth sensors and provide poses of objects that can be used as feedback for controllers, e.g. for visual servoing. Some examples of trackers include model-based object trackers (e.g. [38]) and marker-based trackers like Alvar [9]. The software framework relies on a common interface for the trackers where the output of the tracker is a list of tracking vectors and tracking identification numbers of the objects being tracked. The tracker is also expected to provide a health check on the validity of the tracker output which is used by the state machine logic.

6.2.3 Graphical User Interface

We have also developed a Graphical User Interface (GUI) from which users can trigger events and monitor the robot system and state machine health. The GUI communicates with the state machine through the Robot Operating System (ROS) middleware [189] (Figure 6.4).

6.3 Aerial Manipulator Control

We now describe in detail two of the trajectory tracking controllers implemented on our aerial manipulation system: an acceleration-based controller that relies on roll-pitch-yaw-thrust commands and an MPC controller.



Robot System Status

UAV Status			
Battery Percent: 100.00			
Local x: -0.03	Local y: -0.01	Local z: 1.00	Altitude: 0.00
Roll: 1.79	Pitch: -7.12	Yaw: -175.91	
RC: 0	0	0	0
Acc x: 0.14	Acc y: -0.04	Acc z: 0.05	
Vel x: -0.04	Vel y: -0.00	Vel z: 0.01	
Goal vel x: 0.00	Goal vel y: 0.00	Goal vel z: 0.00	Goal yaw: 0.00
Goal pos x: 0.00	Goal pos y: 0.00	Goal pos z: 0.00	Goal yaw: 0.00
Mass: 2.80	Timestamp: 1410.24		
Quadstate: STANDBY RC SCLOSE G0			
Tracker Status			
Valid: True			
Tracking Vectors:			
7	1.01	0.22	-0.34
28	0.96	-0.01	-0.34
16	0.99	-0.21	-0.34
Arm Status:			
Joint Angles:	-0.92	0.88	
Joint Velocities:	0.00	0.00	
End effector translation:	-0.20	0.00	0.47
End effector RPY:	0.00	-0.04	0.00
CommandStatus: False			
Enabled: True			

UAV Controller Status

Status: Active	

Arm Controller Status

Status: Completed	Reached Goal			
Error Position, Rotation:	0.00	0.00	0.00	0.04

Logic State Machine Status	
Current state:	PrePickState
Last Event without transition:	l
Last Event:	N17pick_place_events4PickE

Figure 6.4: User interface for sending commands and viewing state machine and robot system feedback. The left panel is the portion provided by the framework showing the state machine status. The right panel shows a view from the onboard camera with objects to pick.

6.3.1 Acceleration-based Control

Many off-the-shelf quadrotors come equipped with built-in flight control hardware, where the control interface is limited to higher level commands such as roll-pitch-yaw-thrust or angular rate commands instead of direct motor commands. Here, we describe a controller that can be employed on such systems.

Define the state of the quadrotor as $x = (p, R, v, \omega)$, where $p \in \mathbb{R}^3$ is the position, $R \in SO(3)$ is the attitude, $v \in \mathbb{R}^3$ is the velocity, and $\omega \in \mathbb{R}^3$ is the angular velocity. The autopilot takes as input the desired roll ϕ_d, desired pitch θ_d, desired yaw rate $\dot{\psi}_d$ and a thrust command $u_t \in \mathbb{R}$. It internally runs a feedback loop that controls the rotor velocities to achieve these high-level commands. The aim of the controller is to accurately track a desired reference trajectory in terms of position, velocity, and yaw, where the reference is specified as a smooth trajectory in quadrotor position $p_r \in \mathbb{R}^3$ and quadrotor yaw ψ_r. To achieve this task, we design a controller that computes the desired acceleration $a_d \in \mathbb{R}^3$ based on the error in position $e_p = p_r - p$ and error in velocity $e_v = \dot{p}_r - v$ as

$$a_d = K_p e_p + K_d e_d + a_r, \tag{6.1}$$

where $K_p, K_d \in \mathbb{R}^{3\times3}$ are positive diagonal matrices that act as proportional and derivative gains and $a_r = \ddot{p}_r$ is the feedforward acceleration based on the reference trajectory. The proportional and derivative gains for the x and y axes are selected separately from that of the z-axis gain since the quadrotor dynamics are significantly different along the z-axis.

Next, we compute the roll, pitch, and thrust commands that achieve the desired acceleration a_d. The rotors on the quadrotor are aligned with the body z-axis, which implies the quadrotor can only apply acceleration along this axis. The net acceleration produced by the quadrotor is given by

$$a = R_Z(\psi)R_Y(\theta)R_X(\phi)e_3 u_t - g, \tag{6.2}$$

where ψ, θ, ϕ represent a ZYX Euler parametrization of R, $R_{(\cdot)}$ represents rotation about z, y, and x-axes, $g = [0, 0, -9.81]$ is the gravity vector and $e_3 = [0, 0, 1]^T$ is the body z-axis. Mass does not enter the equation since u_t is a commanded body z-axis acceleration rather than a true thrust force. We solve for the autopilot inputs ϕ, θ, and u_t by setting a as a_d. The desired thrust command is given by

$$u_t = \|a_d + g\|. \tag{6.3}$$

To find the desired roll and pitch, we define the normalized acceleration vector as $\bar{a}_d = (a_d + g)/u_t$. The desired roll and pitch are then given by

$$\phi_d = \arcsin(\bar{a}_d^\top e_1 \sin\psi - \bar{a}_d^\top e_2 \cos\psi), \tag{6.4}$$

$$\theta_d = \arctan\left(\frac{\cos\phi(\bar{a}_d^\top e_1 \cos\psi + \bar{a}_d^\top e_2 \sin\psi)}{\cos\phi \ \ \bar{a}_d^\top e_3}\right). \tag{6.5}$$

The above conversion has a singularity as 90° degrees roll, which is not encountered in our application.

The commanded yaw rate is proportional to the error between the current yaw and desired yaw obtained from the reference trajectory as

$$\dot{\psi}_d = k_\psi(\psi - \psi_r) + \dot{\psi}_r. \tag{6.6}$$

Previous work proves stability for a similar class of trajectory tracking controllers that use PID to compute a desired force and an inner-loop attitude controller to achieve the desired force direction [125].

6.3.2 Model Predictive Controller

The model predictive controller computes the thrust and desired attitude for the quadrotor by solving a trajectory optimization problem. The model predictive controller closely follows the one explained in section 4.1.3.3. We employ the Casadi automatic differentiation library [12] to find the gradients of the dynamics required for the Stagewise Newton method. The MPC optimization for the quadrotor dynamics is able to run at a frequency of 100 Hz on an onboard Intel NUC i5 computer.

6.3.3 Reference Trajectory Generation

The trajectory tracking controllers described above need a reference trajectory that is feasible for the quadrotor to track. When navigating to a waypoint or approaching an object to pick it up, we use a polynomial reference trajectory of degree 9 along each individual axis to ensure the reference derivatives are smooth up to fourth order. The coefficients of the polynomial are found by solving a linear system defined by the boundary conditions of the trajectory, where the initial position and yaw are given by sensors and final position and yaw are given by the user. The rest of the derivatives of the position at the boundaries are set to zero so that the trajectory starts and ends at rest.

6.3.4 Grasping Strategy

Close to the object in the final stage of the picking procedure, we track a trajectory that is constant in the plane parallel to the object, but sinusoidal perpendicular to the object. This results in a periodic "poking" motion. This behavior is desirable since it pushes the end-effector towards the object with the intent of making contact during the first half cycle of the motion, but pulls the end-effector back away from the object if it is misaligned while poking. By pulling away, the robot has the opportunity to correct its attitude and relative position without colliding with the object before the next poking cycle begins.

6.4 Parameter Estimation

6.4.1 Thrust Gain Estimation

The acceleration-based controller relies on the autopilot to achieve the desired thrust, roll, pitch, and yaw rate. The autopilot usually takes as input a normalized thrust command between 0 and 100, where a non-constant scale factor can transform the normalized value to a metric unit of thrust force. The scale factor, called the thrust gain, is constantly changing since it depends on the battery voltage and mass of the quadrotor. To compensate for these effects, a thrust gain estimator computes the mapping between the thrust command and the actual thrust force based on the commanded thrust, the body acceleration vector, and the orientation obtained from the quadrotor. We combine the mass into the thrust gain to directly map the normalized input to gravity compensated acceleration of the quadrotor. The commanded thrust $u \in \mathbb{R}$ maps to a corresponding global acceleration $a \in \mathbb{R}^3$ of the quadrotor as

$$a = k_t R e_3 u + g \tag{6.7}$$

where $k_t \in \mathbb{R}$ is the thrust gain, the orientation of the body is denoted by the rotation matrix R, and the thrust vector is assumed to be pointed towards the body-z direction, i.e. $e_3 = [0, 0, 1]$.

The thrust gain can be obtained from the measured body acceleration $a_b \in \mathbb{R}^3$ and gravity vector as

$$k_t = \frac{1}{u} e_3^T (a_b - R^\top g) \tag{6.8}$$

These measurements can be obtained from the Inertial Measurement Unit (IMU) on the quadrotor. The noise in the IMU measurements is accounted for by using an exponential filter

$$\bar{k}_{t_{i+1}} = (1-\lambda)\bar{k}_{t_i} + \lambda k_{t_i}, \tag{6.9}$$

where $\bar{k}_{t_i}$ is the filtered thrust gain estimate at time index i. By choosing a scale λ between 0 and 1, the thrust gain can be adjusted to change more aggressively, which leads to the quadrotor changing thrust aggressively to compensate for a change in mass.

The estimator is tested on a matrice quadrotor with a 2DOF arm by running the estimator while piloting the drone around and picking objects and dropping them in designated locations. Figure 6.5 qualitatively shows the thrust gain estimated for the quadrotor during one such trial. The positive jumps in the gain denote a package being dropped and a negative jump denotes a package being picked. The thrust gain exhibits an overall downward trend as the battery voltage drops over time.

6.4.2 Euler Angle Bias Estimation

We also found a small difference of approximately 0.5 degrees between the roll and pitch Euler angles reported by the IMU and the angles obtained by inverting the fused body acceleration reported by the IMU a_{acc} as shown in Figure 6.6. The roll and pitch angles corresponding to fused body acceleration are obtained using (6.5) where desired a_d is replaced by the rotated body

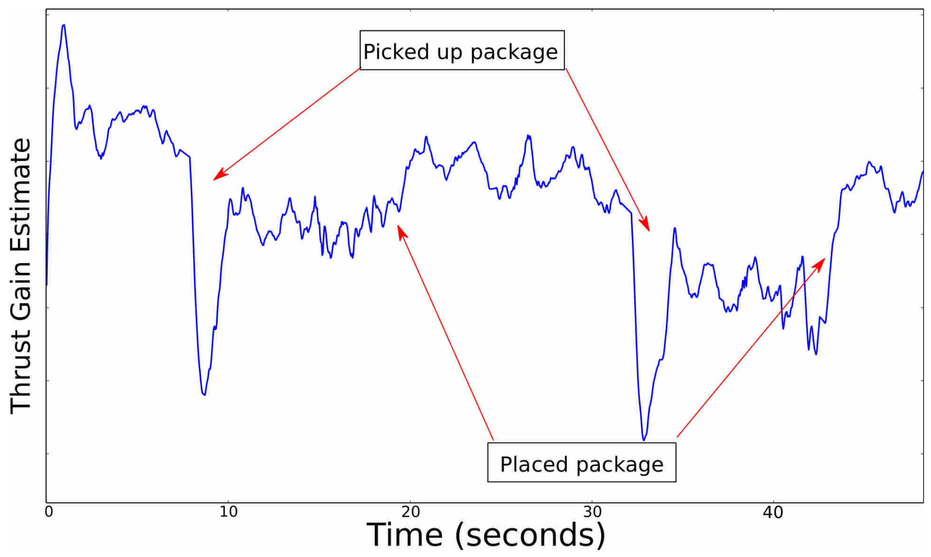


Figure 6.5: Estimate of thrust gain k_t computed from IMU data and expected acceleration.

acceleration reported by the IMU, that is

$$a_{global} = R_Y(\theta)R_X(\phi)a_{acc}, \tag{6.10}$$

where $\bar{a}_{global} = (a_{global} + g)/\|a_{global} + g\|$. To track the reference trajectory, we need to track Euler angles that are consistent with the body acceleration. Hence, we add the difference between the angles $\delta_\phi, \delta_\theta$ to the commanded roll and pitch before sending them to the autopilot, where

$$\delta_\phi = \phi - \phi_{acc}, \;\; \delta_\theta = \theta - \theta_{acc}. \tag{6.11}$$

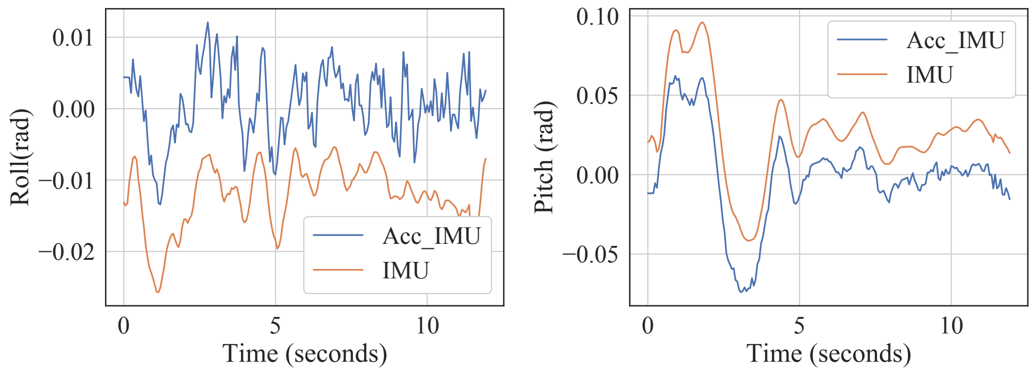


Figure 6.6: Bias in roll and pitch estimated from difference in expected and actual accelerations.

6.5 Hardware

6.5.1 Commercial Off-the-Shelf quadrotor

The aerial manipulation system contains a modified DJI Matrice quadrotor as the base. The quadrotor is equipped with a PointGrey Flea3 camera and an Intel NUCi5 computer, which communicates with the Matrice flight controller over a UART connection.

6.5.2 Motion capture system

Motion capture system is used for position and velocity estimation for the control algorithms. The DJI Guidance sensor suite consisting of 5 stereo cameras is used as a fail-safe in case we lose motion capture during experiments. Although using a motion capture system is restricting the applicability of the system, it provides a good ground truth for comparing our system with other future applications.

6.5.3 Manipulator

6.5.3.1 Custom 2-DOF Arm

Several previous works, like Ghadiok, Goldin, and Ren [64] and Bellicoso et al. [17], develop arms specifically for aerial manipulation, but they typically only grasp objects directly below the robot and cannot reach outside the envelope of the quadrotor. In this work, a light-weight 2-DOF manipulator is used for picking objects outside the envelope of the quadrotor. Dynamixel servos control the manipulator joints which are connected by carbon fiber tubes. The manipulator end-effector is steered using a Cartesian position controller which commands joint velocities to achieve a desired end-effector position. Since the arm is underactuated, the pose of the end effector can only be specified using two translational coordinates.

6.5.3.2 Magnetic Gripper

The arm uses a custom gripper to pick and place objects. Since the position accuracy of the quadrotor is limited to around 2 centimeters, the gripper should be able to pick the object without requiring a high degree of precision. The gripper also needs to be able to pick objects of different sizes and shapes. Existing open-source grippers, such as the Yale OpenHand [137], are too heavy and do not fit the requirements specified above. Our custom gripper shown in Figure 6.7 is composed of four magnets with alternating polarity embedded into a wheel attached to a servo. The magnets are attracted to a mating joint (shown in Figure 6.7) that is attached to any object the user wishes to pick. The mating joint has a pattern of magnets to give the gripper more than one place to attach, thereby increasing the amount of position error it can tolerate while picking. It can tolerate a position error of about 3cm parallel to the surface of the plate and 2cm perpendicular to the plate. Once an object is attached to the gripper, it can be released by rotating the magnet wheel 90° which flips the polarity of the magnets and repels the object. The gripper uses a momentary switch to detect whether it has attached to a mating joint, allowing the onboard computer to know when it has successfully picked up an object. An onboard Teensy microcontroller runs software which sends commands to the servo and receives feedback from the switch. The gripper communication channel connects to a single servo communication bus that runs up the length of the manipulator to the computer.

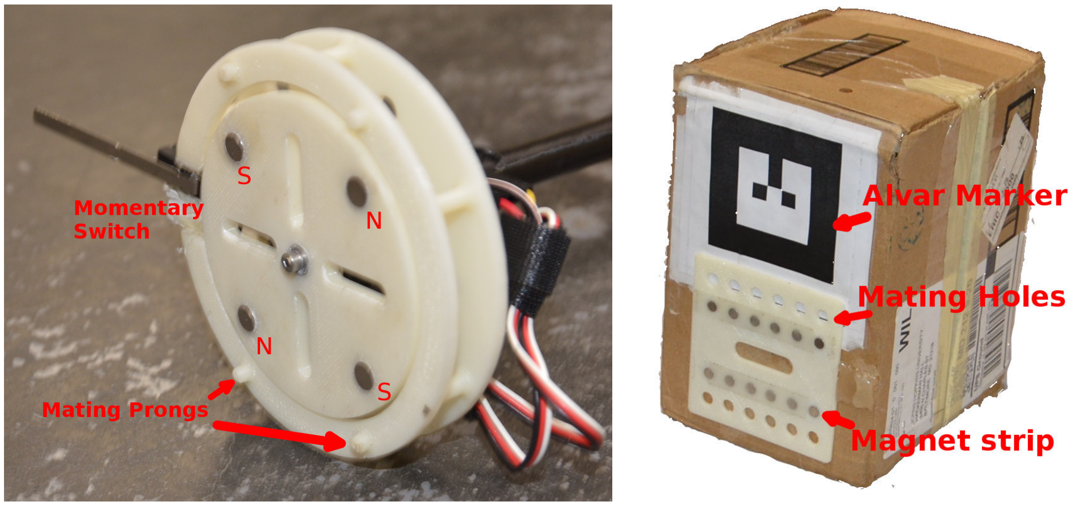


Figure 6.7: The magnetic gripper (left) and a sample package (right) used in our aerial manipulation experiments. The package is instrumented with an AR marker to facilitate tracking and a magnetic mating joint so it can attach to the gripper.

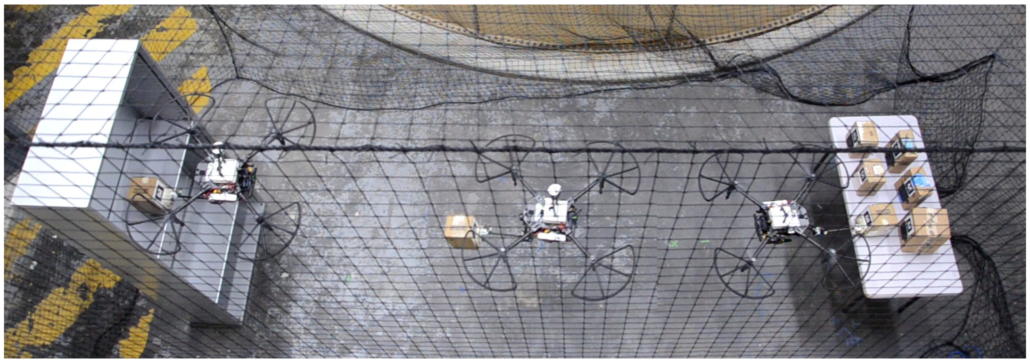


Figure 6.8: Time-lapse of the aerial manipulator picking the object highlighted in yellow from the packaging area (top-left) and placing it on a storage shelf (top-right) in the same trial. The bottom picture shows an overhead view of the pick-and-place procedure.

6.6 Industrial Pick-and-Place Application

The software framework developed in section 6.2 is used to develop an industrial pick-and-place application leveraging the aerial manipulation platform described in section 6.5.

6.6.1 Experiment setup

The goal of the application is to sort packages from a packaging area (table) and transport them to corresponding storage area (shelf). The package transportation capability can be useful, for instance, in package fulfillment centers or for remote object transport in radioactive environments.

The packages are tagged with AR markers [9] and have an attached mating joint that connects to the gripper described in section 6.5.3.2. Each package has a corresponding destination marker ID where the object is placed. Figure 6.8 shows a timeline of the quadrotor picking and transporting packages to their corresponding storage spaces. The packages have masses between 120g and 170g. The mass of the package is limited by the arm capacity (200g) and the quadrotor payload capacity (500g).

State Machine

Figure 6.9 shows a simplified illustration of the finite state machine for the pick and place application. There are two different logical loops in the diagram.

The first is the regular logic loop starting from "Waiting to Pick" state. During this cycle, the quadrotor automatically detects the closest available package in the workspace, picks up the package, determines the storage location based on the marker ID of the object picked up, uses visual servoing using on-board camera to navigate to a marked shelf, places the package on the shelf, and returns to a start position with the packages in view. This process is repeated indefinitely assuming new packages appear continuously in the packaging area.

Various system components could fail throughout the pick-and-place process, but the implemented state machine accounts for such failures through a second loop known as the fault-recovery loop. For example, during picking, the arm could block the marker from the camera, resulting in a tracking loss. Instead of just aborting and waiting for human input, the system instead back-tracks to its prior position and re-attempts the picking process. Other failure modes include failing to pick the object within a specified timeout. Recovery state transitions are shown in red in Figure 6.9.

In addition to automatic recovery, sometimes during the experiment, a user intervention is necessary. A list of such failures encountered during operations is listed in Table 6.4. The state machine ensures the system is safe under these failure modes by switching to hovering and relying on internal controller to stably hover in place until the user is ready to intervene. Furthermore, the state machine accepts manual override from a safety pilot to abort any action safely. The state machine recognizes the intervention and aborts any active controllers running on the machine. Therefore, the user can resume picking operation after rectifying the error and disabling the override sent.

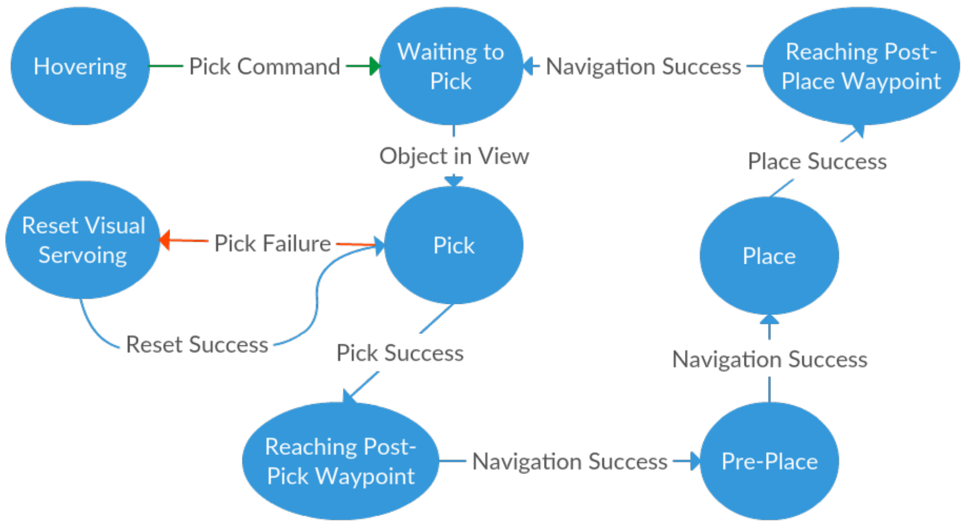


Figure 6.9: Part of the state machine for picking and placing a package. The recovery actions are red and user actions are green. The user can also abort from any other state back to hovering if manual intervention is desired.

6.6.2 Results

Figure 6.8 shows a timeline of the pick-and-place task, where the quadrotor picks up a package from the table and places it in a shelf. The media attachments associated with this work demonstrate the complete pick-and-place

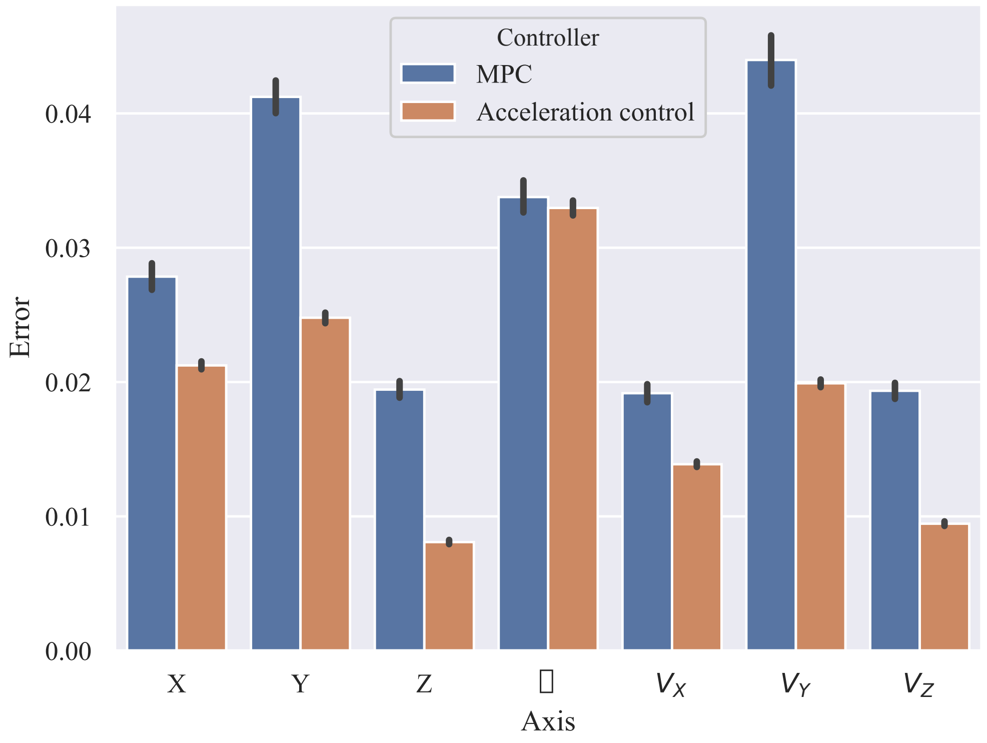


Figure 6.10: Mean absolute errors along x, y, z (meters), and yaw ψ axes (radians) and translational velocities (meters/second) for MPC and acceleration-based controller. The black lines show the 95% confidence interval obtained using bootstrapping.

task where the quadrotor sorts multiple packages into the top and bottom shelves without any manual interruptions.

We quantified the ability of the quadrotor to perform a successful pick operation over 101 trials of picking and placing. The acceleration-based controller is used for these trials since it was easier to tune and performed slightly better than MPC at the picking task. Figure 6.10 compares the mean absolute errors along translational positions, velocities, and yaw angle for each controller. Both the MPC controller and acceleration-based controller were able to track reference trajectories within 5 centimeters RMS error, but the acceleration-based controller with more extensive gain tuning reduced the

RMS errors by two centimeters.

Table 6.3 shows the mean trajectory tracking errors and pick times during the trials. The aerial manipulator was able to pick the object successfully 80% of the time without the ability to detect system faults. The system's pick success rate increased to 85% when it is was able to automatically recognize failure to pick an object and could retry and re-pick the object in a future attempt. We also achieved a mean absolute error of less than 3 centimeters in all translational axis and less than 2 centimeters/second in velocity. Figure 6.11 shows a histogram of pickup times and total time for pick and place of an object over different trials. The majority of pickup times vary from 6 seconds to 16 seconds, while the total pick-and-place time for one box varies from 30 seconds to 40 seconds in most cases.

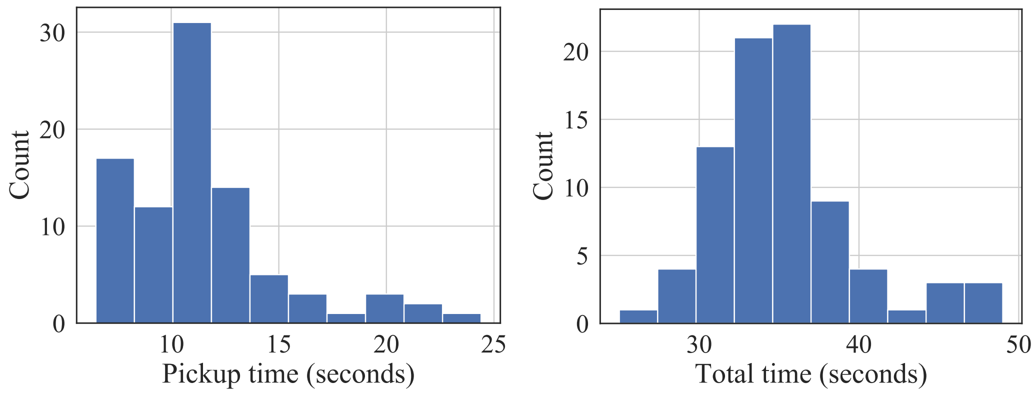


Figure 6.11: Histogram of pickup and total time for placing one box using the pick place state machine

Pick Success Rate	91/101
Overall Success Rate	85/101
Min Pick Time	6.5 seconds
Mean Pick Time	11.5 seconds
Max Pick Time	25 seconds
Mean Absolute Error x	2.1cm
Mean Absolute Error y	2.5cm
Mean Absolute Error z	1 cm
Mean Absolute Error ψ	0.03 rad

Table 6.3: Pick success rate, pick time statistics, and error in quadrotor position and yaw for picking an object

Failure Mode	Number of Failures
Object misplaced in shelf while placing	1
Gripper failed to hold onto object	1
Lost motion capture while gripping	1
Controller failed after multiple retries	3
Proximity sensor failed to detect object	3
Object went out of workspace	3
Camera stops responding due to driver errors	3

Table 6.4: Failure modes during pick-and-place trials

6.7 Conclusion

This chapter developed an aerial manipulation system using a commercial quadrotor, a custom arm and end-effector, and a new software framework for aerial autonomy capable of fault-tolerant industrial pick-and-place tasks. While failure detection and system health monitoring increased the robustness of the system, more robust hardware and environment-adaptive manipulation are necessary to further reduce the failure modes shown in Table 6.4 and drive the system toward 100% reliability. Future work will integrate advanced adaptive models for the quadrotor and the arm that explicitly take into account their coupled dynamics in order to reduce position control error in MPC methods. New grippers that do not require custom attachments on the package will also be designed to make the system more widely applicable. Finally, while we were able to demonstrate reliable and relatively efficient operation, the overall speed and agility of the robot can be further improved. Achieving extreme agility without sacrificing reliability remains a central challenge yet to be solved.

Chapter 7

Other Related Work

This chapter introduces two related methods for controlling underactuated and non-holonomic robotic systems. First approach extends a standard gyroscopic obstacle avoidance controller for an underactuated system and prove it's Lyapunov stability. Simulations show that the gyroscopic controller is able to stabilize two underactuated systems i.e a quadrotor and a satellite to a goal state in the presence of multiple obstacles blocking the robot. Second approach proposes applying NMPC optimization to a high fidelity physics engine based dynamic model of the robot. We show that using such near global optimization methods such as Cross-Entropy search method [103], we can find optimal paths that avoid obstacles.

7.1 A Stabilizing Gyroscopic Obstacle Avoidance Controller for Underactuated Systems

7.1.1 Introduction

In this chapter, we tackle the problem of navigating a cluttered environment using a underactuated dynamic robotic system. One of the key challenges is dealing with the inability of the system to instantaneously produce a force in any desired direction. These types of systems are common in robotics and include quadrotors, satellites, and underwater vehicles. For instance, quadrotors are becoming important for a variety of applications which require robust navigation, such as search and rescue [68], mapping [208], and package delivery [184]. Small satellites, such as cubesats [203], are enabling low-cost testbeds for applications like formation flying [28] and other autonomous operations [110]. We develop a gyroscopic obstacle avoidance controller for this particular class of underactuated systems, namely rigid bodies with controls given by body torques and a thrust force along some body-fixed axis.

A number of control techniques have been developed for such types of vehicles. These include deterministic feedback linearizing controllers for quadrotors [8, 5, 123, 226] as well as Lyapunov-stable controllers in the presence of bounded external disturbances [108, 140, 23, 190].

In addition to standard point stabilization, requiring provably stable obstacle avoidance significantly complicates the control design. While a backstepping obstacle avoidance controller has been proposed in Geng, Shuai, and Hu [62], it focused on two schemes, namely a mass point model and a safety ball model to generate waypoints away from an obstacle. The proposed controller, therefore, does not explicitly include obstacle constraints, and no proof of obstacle-aware convergence is available. Traditionally, obstacle avoidance for fully actuated systems has been considered through gyroscopic avoidance [31] and gradient vector field [194] approaches. The gradient vector field approach generates an vector field obtained as the gradient of a navigation function with a global minimum at the goal and maxima at the obstacles, but the design of such a navigation function is nontrivial. The navigation field approach has been used for obstacle avoidance of quadrotors [33, 25]. Unlike the globally stable potential field methods, the dynamic window approach [166] is a local method that merges a Model Predictive Control (MPC) approach and potential field methods to find a control trajectory in the accessible space that maximizes a utility function.

In contrast to requiring a potentially complex nonlinear potential function, the gyroscopic avoidance approach handles obstacles by applying a steering force to the robot without increasing the Lyapunov energy of the system [31]. The obstacle-avoiding force, therefore, does not affect the Lyapunov stability of the controller and ensures the method is semi-globally convergent to the goal state in the presence of unknown convex obstacles. Gyroscopic avoidance has been successfully applied to create flocking behaviour in a multi-agent system [32] and to control an Unmanned Ground Vehicle (UGV) [225]. A gyroscopic force added to a potential field approach was applied to quadrotor swarm formation in Min, Sun, and Niu [155], but only a simplified kinematic model for the quadrotor was considered.

We extend the gyroscopic avoidance approach [31](originally developed for setpoint control of fully actuated systems) to underactuated dynamical systems in 3D workspaces through a backstepping technique. To ensure convergence in the presence of multiple obstacles, a novel obstacle-avoiding steering function has been designed to enable smooth transitions between colliding and non-colliding directions of motions. Furthermore, to ensure stability even when the system has a finite obstacle detection radius, a smooth obstacle control gain is employed. Two types of 3D obstacles are considered: cylinders and spheres. Many real world obstacles can be modeled using a combination of these primitives.

We demonstrate the ability of the method to perform setpoint control while avoiding obstacles in two challenging simulated scenarios. The controller is first employed on a quadrotor and shown to converge to a goal position while avoiding a forest of trees modeled as cylinders with spherical canopies. A similar example involving a nanosatellite is shown to converge to a goal position while avoiding space debris modeled as spheres. To the best of the authors' knowledge, this is the first controller providing convergence for these types of underactuated systems in complex scenarios.

The rest of the chapter is organized as follows. In section 7.1.2 we specify the dynamics of the class of underactuated systems considered. In section 7.1.3, a desired gyroscopic controller is designed for the translational dynamics. This controller is extended to the class of underactuated systems through backstepping in section 7.1.4. Next, the design of gyroscopic obstacle avoidance gains specific to cylindrical and spherical obstacles is specified in section

7.1.5. Finally, simulations of a quadrotor and nanosatellite in non-trivial scenarios are shown in section 7.1.6. The proof for stable collision avoidance is derived in Appendix 7.1.8.

7.1.2 Dynamics of Underactuated systems

We consider underactuated systems modeled as rigid bodies with position $p \in \mathbb{R}^3$ and velocity $\dot{p}$ in a fixed inertial frame, orientation matrix $R \in SO(3)$ and body-frame angular velocity $\omega \in \mathbb{R}^3$. The control inputs for the system are the body torques $\tau \in \mathbb{R}^3$ and thrust force $u \in \mathbb{R}$ in some known body-fixed direction $e \in \mathbb{R}^3$. The system is subject to known external forces given by $f \in \mathbb{R}^3$ and no external torques. The dynamics is

$$m\ddot{p} = Reu + f, \tag{7.1}$$

$$\dot{R} = R\widehat{\omega}, \tag{7.2}$$

$$J\dot{\omega} = J\omega \times \omega + \tau, \tag{7.3}$$

where m is the mass and J is the rotational inertia.

Our goal is to design a Lyapunov-stable controller achieving a given desired goal position p_d with zero velocity $\dot{p}_d = 0$ while avoiding obstacles. To accomplish this, we first design a gyroscopic obstacle avoidance controller for the translational dynamics of the underactuated system. The resulting "desired" control forces for this controller cannot be directly achieved due to underactuation. Therefore, we perform a backstepping procedure which closes the loop in stages and ultimately achieves stability. We next describe the translational and gyroscopic parts of the controller derivation.

7.1.3 Gyroscopic Avoidance

We first design a gyroscopic avoidance controller for the position coordinates. Let $g \in \mathbb{R}^3$ denote the translational input force. For clarity, let the system's translational state combining position and velocity be denoted by $x = [p^T, \dot{p}^T]^T$. The translational dynamics is then given by

$$\dot{x} = Ax + B(g + f), \tag{7.4}$$

$$A = \begin{bmatrix} 0 & I_{3\times3} \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} I_{3\times3} \end{bmatrix}. \tag{7.5}$$

The underactuated dynamics considered in (7.1-7.3) is an extension of this subsystem, where the control force is given as the thrust vector $g = Reu$ and the thrust direction Re is controlled by the rotational dynamics of the system.

The design of the controller starts with defining the error $z_0 \in \mathbb{R}^6$ between the state x and the desired state $x_d = [p_d^T, 0^T]^T$, given by

$$z_0 = x - x_d.$$

For a standard linear system considered in (7.4-7.5), a Lyapunov stable feedback control law can be achieved using a desired force $g_d \in \mathbb{R}^3$ given by

$$g_d = -Kz_o - f, \tag{7.6}$$

$$K = [K_p,\ K_v], \tag{7.7}$$

where $K_p, K_v \in \mathbb{R}^{3\times 3}$ are positive definite matrices corresponding to proportional and derivative gains, respectively.

Gyroscopic avoidance control is equivalent to adding force terms which steer the system around the obstacles. This is accomplished by adding a desired force perpendicular to both the current velocity and the steering axis of the obstacle. The desired force is then augmented according to

$$g_d = -Kz_0 - f - G(x)\dot{p}, \tag{7.8}$$

where the matrix $G(x) \in \mathbb{R}^{3\times 3}$ is skew-symmetric, i.e. it instantaneously rotates the velocity. Equivalently, the *steering force* $G(x)\dot{p}$ can be regarded as being perpendicular to the velocity of the robot.

For stability analysis it will be useful to define the storage function

$$V_0 = \frac{1}{2} z_0^T P z_0, \tag{7.9}$$

where

$$P = \begin{bmatrix} K_p & 0 \\ 0 & mI_{3\times 3} \end{bmatrix}. \tag{7.10}$$

We then have

$$\dot{V}_0 = \frac{1}{2} z_0^T P(Ax + B(g+f)) + \frac{1}{2}(Ax + B(g+f))^T P z_0.$$

If the system is fully actuated, the input force g can be set to the desired control force g_d, and it would be possible to show asymptotic stability. Yet, the input force for an underactuated system is restricted only along the single body direction Re and cannot be set to the desired force g_d. Hence, the above controller is extended to the full underactuated system dynamics using a backstepping procedure as described next.

7.1.4 Backstepping Procedure

We next close the loop in stages using backstepping. The controller designed in (7.6) requires the control force on the system g to be equal to a desired control force g_d that stabilizes the system. This desired force g_d cannot be achieved directly for underactuated systems, since the control force can only be applied along some known body direction Re. Instead, the difference between the applied force g and desired force g_d is considered as an error $z_1 \in \mathbb{R}^3$ defined by

$$z_1 = g - g_d$$

to be further suppressed in the backstepping procedure. With this definition we have

$$\dot{V}_0 = -\frac{1}{2} z_0 Q z_0 + (B^T P z_0)^T (g - g_d),$$

where

$$Q = 2 \begin{bmatrix} 0 & 0 \\ 0 & -K_v \end{bmatrix}. \tag{7.11}$$

Next, a new Lyapunov candidate is defined which includes z_1 as

$$V_1 = V_0 + \frac{1}{2} z_1^T z_1.$$

with time-derivative given by

$$\dot{V}_1 = -\frac{1}{2}z_0 Q z_0 + z_1^T(\dot{g} - \dot{g}_d + B^T P z_0),$$

which can be expressed as

$$\dot{V}_1 = -\frac{1}{2}z_0 Q z_0 - k_{z_1} z_1^T z_1 + z_1^T z_2,$$

where $z_2 = \dot{g} - a_d$ and $a_d = \dot{g}_d - B^T P z_0 - k_{z_1} z_1$. The variable a_d is a desired value for $\dot{g}$ which cannot be instantaneously achieved by the underactuated system. Thus, continuing the backstepping procedure, a new Lyapunov candidate which includes the error between the desired and actual values of $\dot{g}$ is given as

$$V_2 = V_1 + \frac{1}{2}z_2^T z_2,$$

with derivative

$$\dot{V}_2 = -\frac{1}{2}z_0 Q z_0 - k_{z_1} z_1^T z_1 + z_2^T(\ddot{g} - \dot{a}_d + z_1).$$

The desired value of $\ddot{g}$ which ensures $\dot{V}_2$ is negative definite is denoted as b_d, and is computed as

$$\begin{aligned} b_d &= \dot{a}_d - z_1 - k_{z_2} z_2 \\ &= \ddot{g}_d - B^T P \dot{z}_0 - k_{z_1} \dot{z}_1 - z_1 - k_{z_2} z_2, \end{aligned}$$

where

$$\ddot{g}_d = -[0 \;\; \ddot{G}(x)] z_0 - 2[0 \;\; \dot{G}(x)] \dot{z}_0 - K \ddot{z}_0$$

and

$$\ddot{z}_0 = A^2x + AB(f+g) + B\dot{g}.$$

For the underactuated system, the derivative $\ddot{g}$ can be expanded as

$$\ddot{g} = R[e\ddot{u} + \hat{\dot{\omega}}eu + 2\hat{\omega}e\dot{u} + \hat{\omega}^2eu].$$

Finally, the control inputs $\tau, \ddot{u}$ can be chosen to satisfy the desired b_d by setting

$$\tau = \mathbb{J}[e \times (R^Tb_d - \hat{\omega}^2eu - 2\hat{\omega}\dot{u})/u] - \mathbb{J}\omega \times \omega \tag{7.12}$$

$$\ddot{u} = e^T(R^Tb_d - \omega^2eu - 2\hat{\omega}\dot{u}). \tag{7.13}$$

Note that this controller does not directly control the thrust force u, but instead controls $\ddot{u}$. The state of quadrotor system is extended by $u, \dot{u}$ to account for this. The controller for torque τ shown in (7.12) has a singularity when the thrust force goes to zero. This is not a problem in practice since the vehicle always produces a positive force u while navigating.

Proof of Stability:

Let the error state for the system be given by $z = (z_0, z_1, z_2)$. Using the control law described in (7.12), the derivative of Lyapunov function V_2 is evaluated as

$$V_2 = \frac{1}{2}(z_0^T P z_0 + z_1^T z_1 + z_2^T z_2) \tag{7.14}$$

$$\dot{V}_2 = -\frac{1}{2} z_0^T Q z_0 - k_{z_1} z_1^T z_1 - k_{z_2} z_2^T z_2 = \frac{1}{2} z^T K z \tag{7.15}$$

$$\text{with } K = \begin{bmatrix} Q & 0 & 0 \\ 0 & -2k_{z_1} & 0 \\ 0 & 0 & -2k_{z_2} \end{bmatrix}, \tag{7.16}$$

where the matrix K is negative semidefinite. Based on Lasalle's invariance we know we will end up in the largest invariant set corresponding to $\dot{V}_2 = 0$.

Let us find the largest invariant set [99] with respect to the quadrotor dynamics. When $\dot{V}_2 = 0$, the scaled error $z_0^T Q z_0 = 0$. Based on the form of Q from equation (7.11), the scaled error goes to zero only if velocity is zero for all time $\dot{p} = 0, \forall t > t_0$, where t_0 is the time the system enters the invariant set. This implies that the resulting acceleration is zero during the time the robot is in the invariant set ($\ddot{p} = 0$). The acceleration of the system goes to zero only when the translational force g compensates the external force $(g = -f)$ according to equations 7.4 and 7.5. Since we are in the set $\dot{V}_2 = 0$, $z_1 = 0$ and therefore $g = g_d$. This implies that the desired translational force is equal to the negative of external force $g_d = -f$. Equation (7.8) implies that $Kz_0 = 0$. Since K is full rank, $z_0 = 0$. To stay in the set $\dot{V}_2 = 0$, the system should satisfy $z_0 = 0$ which implies that it should reach the goal position. Therefore the largest invariant set inside $\dot{V}_2 = 0$ is given by $[z_0, z_1, z_2] = 0$.

Assuming V_2 is a C^1 smooth function and assuming the initial error state $z(t = 0)$, controls $\tau(t), u(t)$, and states $x(t)$ are bounded, the system will asymptotically converge to $z = 0$ according to Lasalle's invariance principle. The boundedness of the states and controls is ensured only if the robot does not collide with an obstacle. The proof for collision avoidance is provided in section 7.1.8 which guarantees boundedness of the control inputs. The smoothness of $\dot{V}_2$ is ensured by a smooth control law, which implies the steering forces must be C^2 smooth. The design of proper steering forces is discussed in the following section.

7.1.5 Obstacle Avoidance Coefficients

In the backstepping controller shown in (7.12), the specific form of gyroscopic avoidance matrix $G(x)$ is required to be C^2 smooth.

The obstacle avoidance matrix $G(x)$ is composed of a sum of obstacle avoidance matrices $G_i(x)$ corresponding to individual obstacles. Each individual obstacle avoidance matrix is further decomposed into two components: an angular obstacle avoidance gain $k_1(\theta_i)$ and a radial obstacle avoidance gain $k_2(d_i)$. The obstacle avoidance matrix is also designed to induce a rotation around the obstacle steering axis $e_i(x)$. In summary, we have

$$G(x) = \sum G_i(x), \quad G_i(x) = k_1(\theta_i) k_2(d_i) \hat{e}_i(x). \tag{7.17}$$

where the hat operator $\hat{\cdot}$ maps the steering axis in $\mathbb{R}^3$ to a skew symmetric matrix in $\mathbb{R}^{3x3}$ as

$$\hat{e} = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix}.$$

The angular obstacle avoidance gain reduces the magnitude of steering force as the robot heads away from the obstacle and the radial obstacle avoidance gain forces the magnitude of steering force to be inversely proportional to the distance between the robot and obstacle. In addition, the gain should be negligible beyond a finite detection radius from the obstacle. Since the second order derivatives of G are used in the backstepping control law (7.12), the obstacle avoidance gains and steering axis direction should be C^2 smooth functions.

The obstacles considered in this work are either cylinders or spheres. The choice of obstacle avoidance gain and steering axis for the two obstacles are discussed next.

Cylinders

A cylinder is specified by its major axis direction unit vector a, radius r, and a point on the major axis o_p. The cylinder obstacles are assumed to have infinite length. The steering axis $e(x)$ for a cylindrical obstacle is chosen along the major axis direction a. This applies a steering correction around the major axis of the cylinder. There is an ambiguity in the sign of the major axis of the cylinder, which corresponds to steering to the right or left of the cylinder. The sign of the steering axis is determined according to (7.18-7.21). According to this approach, the robot avoids the cylinder by steering right if it is heading to the right of cylinder, and the opposite when heading to the left of the cylinder.

$$v = \dot{p} - (a^T \dot{p})a \tag{7.18}$$

$$\Delta p = o_p - p \tag{7.19}$$

$$d = \Delta p - (a^T \Delta p)a \tag{7.20}$$

$$e(x) = \text{sign}(a^T (d \times v))a \tag{7.21}$$

The velocity v is the projected robot velocity onto the plane perpendicular to the major axis of the cylinder $a^{\perp}$. Similarly, the displacement vector d is the projection of the vector connecting the robot center to the point on the major axis of the cylinder onto the plane $a^{\perp}$.

The sign function is not differentiable, hence it is not suitable for a backstepping control law. A scaled sigmoid function which is a smooth approximation of the sign function is chosen instead. The smooth steering axis is given by:

$$e(x) = [2S(a^T (d \times v)) - 1]a,$$

$$\text{where } S(t) = \frac{1}{1 + e^{-kt}}.$$

The angular obstacle avoidance gain is given by

$$k_1(\theta) = e^{k_{att}(\theta - 1)}, \tag{7.22}$$

$$\text{where } \theta = \frac{d^T v}{\|d\| \|v + \lambda a\|}. \tag{7.23}$$

The variable θ is the cosine of the angle between the projected velocity v and the projected displacement vector d, and $k_{att} \in \mathbb{R}^+$ adjusts the sensitivity of the steering force to the robot's heading. When the magnitude of the projected velocity approaches zero, the angle between d and v is undefined. Therefore, a velocity vector in the null space of projected velocity is added to the projected velocity in (7.23) to avoid this singularity. This regularizing term also ensures that the derivatives of $k_1(\theta)$ remain bounded. Note that the standard regularization of θ by adding a constant to denominator as in $\|d\|\|v\| + \lambda$ does not regularize the derivatives.

The specific form of the radial obstacle avoidance gain is given as:

$$k_2(d) = k_{obs} \frac{S(r_d + r - \|d\| - \epsilon)}{\|d\| - r}. \tag{7.24}$$

A sigmoid function is used as an smooth approximation of the step function to suppress the radial gain beyond the finite detection radius r_d. Note that the sigmoid function is used for convenience, and a spline function with smooth derivatives until second order can also be used. The value of ϵ is adjusted such that the radial obstacle avoidance gain at the detection radius is negligible. The gain k_{obs} scales the steering effort. An appropriate value for k_{obs} can guarantee obstacle avoidance as discussed in appendix 7.1.5.

The complete obstacle avoidance matrix for a cylindrical obstacle is written as

$$G(x) =$$

$$k_{obs} \frac{e^{k_{att}(\theta - 1)}}{\|d\| - r} S(r_d + r - \|d\| - \epsilon) S(a^T(v \times d))\hat{a}$$

Spheres

A sphere is specified by it's center o_p and radius r. Unlike a cylinder, the steering axis for a sphere is not constant, and is chosen based on the robot velocity and displacement vector from the robot to the center of the sphere. The form of steering axis chosen is given by

$$e(x) = \frac{d \times v}{\|d \times v\|} \tag{7.25}$$

$$\text{where } v = \dot{p}, \tag{7.26}$$

$$d = o_p - p. \tag{7.27}$$

When the robot is heading towards the obstacle, the cross product of $d \times v$ goes to zero. This creates unbounded derivatives of the steering axis. To avoid this issue, a regularized value of norm of the cross product is used. The norm of the cross product can be written as

$$\|d \times v\| = \sqrt{\|d\|^2\|v\|^2 - (d^T v)^2}.$$

The regularized value can then be formulated as

$$\|d \times v\|_{reg} = \sqrt{\|d\|^2\|v\|^2(1 + k_{reg}) - (d^T v)^2},$$

where the gain k_{reg} ensures that the norm $\|d \times v\|_{reg}$ is non-zero even when the robot is heading towards the obstacle.

The radial obstacle avoidance gain $k_2(d)$ is the same as that of cylinder obstacle. The steering gain chosen for a spherical obstacle is given by

$$k_1(\theta) = e^{k_{att}(\theta - 1)},$$

$$\text{where } \theta = \frac{d^T v}{\|d\| \|v\|}.$$

The steering obstacle avoidance gain and the steering axis are not defined when the robot velocity is zero. This singularity is avoided by setting G to zero when the velocity of the robot is below a threshold δ. The complete obstacle avoidance matrix G for a sphere is written as

if $\|v\| > \delta$

$$G(x) = k_{obs} \frac{e^{k_{att}(\theta-1)}}{\|d\| - r} \frac{S(r_d + r - \|d\| - \epsilon)}{\|d \times v\|_{reg}} \widehat{d \times v}$$

else

$$G(x) = 0.$$

7.1.6 Numerical Simulations

The backstepping controller designed in section 7.1.4 is tested on a quadrotor and a nanosatellite to perform setpoint tracking in simulation. The underactuated dynamics in (7.1) has been discretized using a semi-implicit scheme [104] and integrated using Euler integration.

A finite detection radius of 3 meters is applied to the obstacles. The backstepping parameters have been chosen to ensure the controls are bounded and the system converges smoothly to the goal. The position gains have also been selected carefully based on the distance to the set point goal to ensure optimal performance.

7.1.6.1 Quadrotor in a Dense Forest

A standard quadrotor model is used, with the thrust aligned with the body-fixed z-axis of the system and with gravity as the only external force, i.e $f = (0, 0, -9.81m)$. The mass and moment of inertia of the quadrotor are chosen as 0.5 Kg a diagonal matrix $J = diag([.003, .003, .005])$ respectively. The obstacle scene used for testing is that of a forest with tree obstacles as shown in Fig. 7.1. Each tree obstacle is constructed of a cylindrical trunk and a spherical canopy. A total of 23 trees are generated on a grid spanning a 20m$\times$20m region. The trees are perturbed randomly from the grid centers. The goal of the controller is to reach a desired position shown in Fig. 7.1 starting from the opposite side of the grid.

One can observe that the quadrotor smoothly reaches the goal while avoiding obstacles as illustrated in Fig. 7.1. The state, control, and Lyapunov energy history of the quadrotor trajectory are shown in Fig. 7.2. Note that the Lyapunov energy function asymptotically approaches zero with no discontinuities, even though the obstacle detection radius of the system is finite.

7.1.6.2 Satellite among Space Debris

For the second example, consider a nanosatellite equipped with an attitude control system and a single thruster. The satellite is placed in an environment of space debris modeled as spheres, and no external forces are included in the

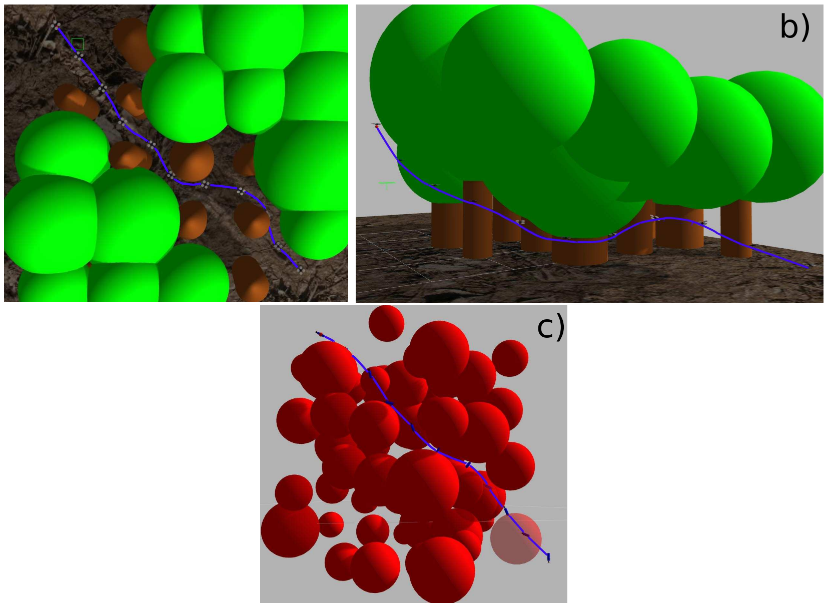


Figure 7.1: The figures (a) and (b) show a quadrotor navigating through a dense cluster of trees using the proposed controller with a finite obstacle detection radius. Figure (a) shows the top view of the quadrotor, where the obstructing obstacles have been removed for a clear view of the trajectory. Figure (b) shows a side view of the same trajectory swooping under the canopies to reach the goal. A satellite navigating through space debris using the proposed controller is illustrated in (c). The thrust vector of the satellite is shown by a pointed cone at the bottom of the satellite.

simulation. The satellite is required to navigate to a desired position while avoiding the obstacles. A total of 48 obstacles are generated using a cuboid grid in a cubic space of 9m$\times$9m$\times$9m. The obstacles are randomly perturbed around the grid centers. The goal of the cubesat is to reach a goal position shown in 7.1 starting from an initial position using the proposed controller.

Similarly to the quadrotor scenario, the satellite is able to stabilize to the goal while avoiding obstacles as demonstrated in Fig. 7.1. A detailed state, control, and Lyapunov energy history of the satellite is shown in Fig. 7.3. The Lyapunov energy function asymptotically approaches zero even as new obstacles enter or leave the detection radius of the system.

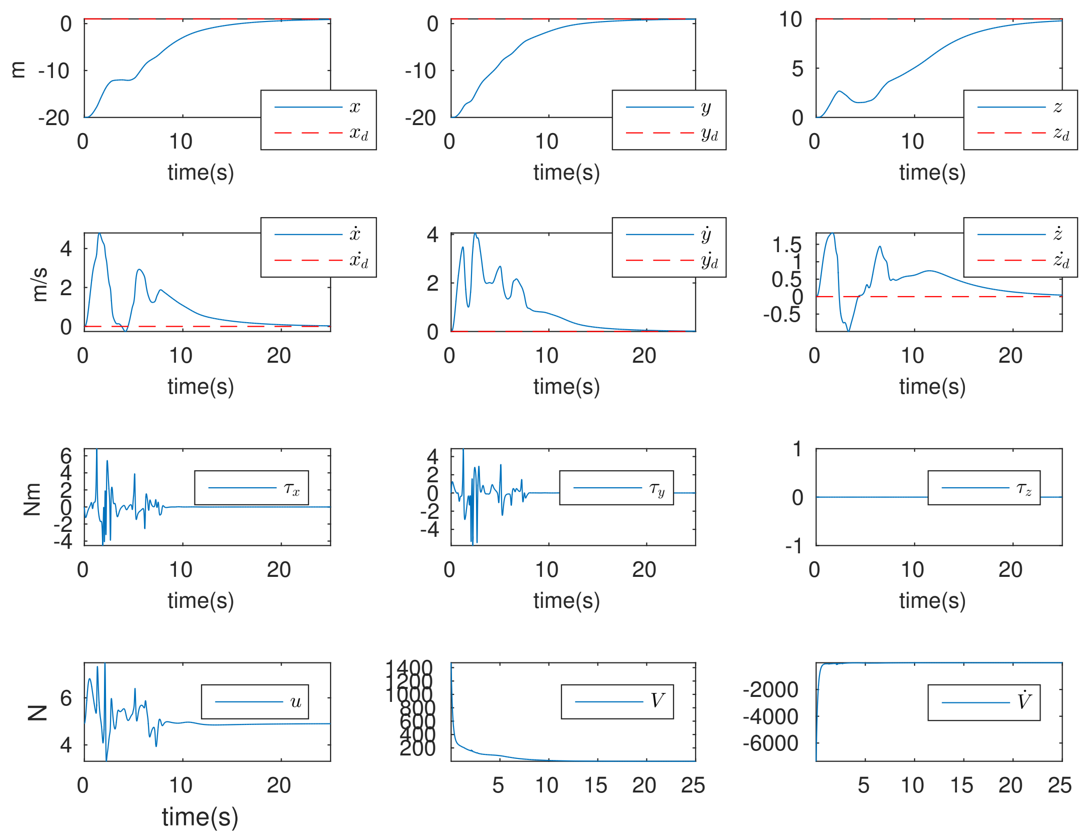


Figure 7.2: History of the quadrotor position, velocity, controls, and Lyapunov function V for the quadrotor simulation

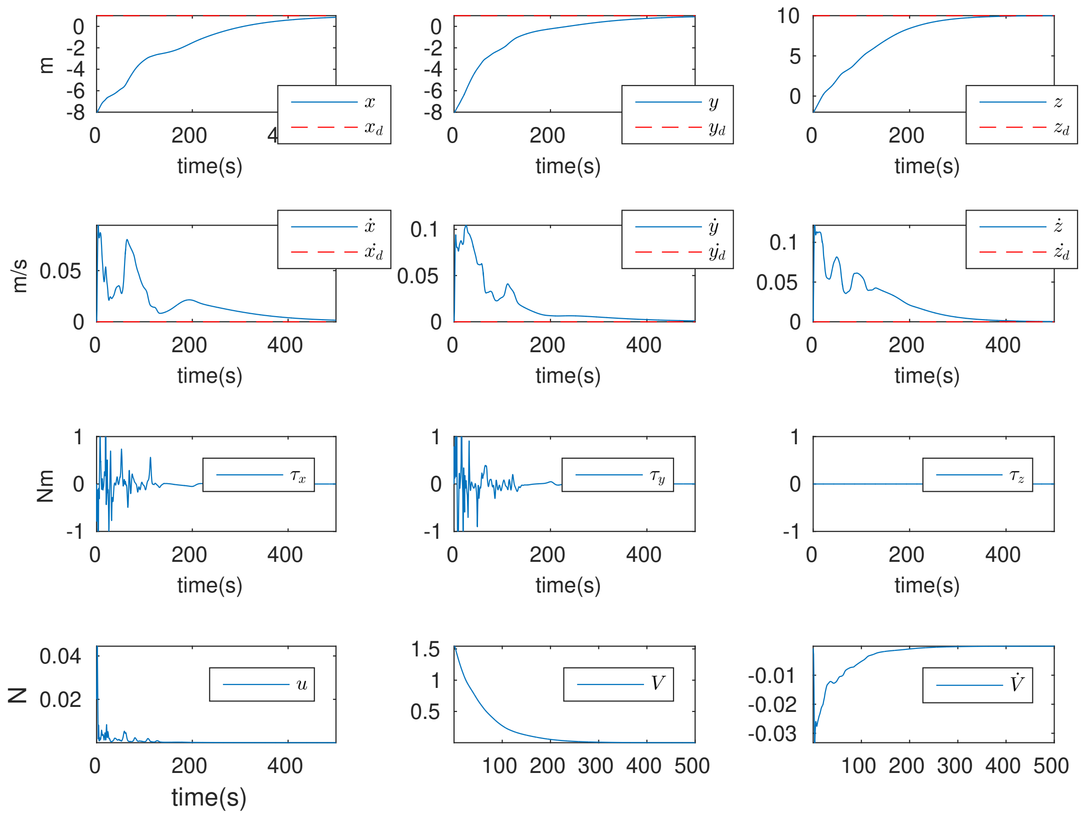


Figure 7.3: History of the nanosatellite position, velocity, controls, and Lyapunov function V for satellite simulation

7.1.7 Conclusions

A backstepping Lyapunov stable controller using gyroscopic obstacle avoidance has been designed for underactuated systems. The ability of the controller to handle finite detection radius and the underactuated dynamics in the presence of obstacles has been shown in theory, and simulations verified the capability of the controller to avoid obstacles and converge to a goal in complex scenarios. Furthermore, the obstacle coefficient design approach employed in this work can be extended to include new primitives such as finite length cylinders, finite planes, and ellipsoids. The backstepping procedure can also be extended to stabilize the system under bounded external disturbances as explained in [108]. Future work will concentrate on implementing the gyroscopic obstacle avoidance controller on a real system and showing that the convergence guarantees hold.

7.1.8 Appendix

Proof for Obstacle Avoidance

In this section, the appropriate choice of scaling gain on the distance obstacle avoidance k_{obs} shown in (7.24) required to guarantee obstacle avoidance of the system is presented. Several assumptions are required for finding the gain. To start with, it is assumed that there is a single obstacle in the environment. Further, the gain matrices K_p and K_v used in the desired force g_d in (7.8) are assumed to be constant matrices as in $k_p I_{3\times3}, k_v I_{3\times3}$, and we set $m = 1$ for simplicity. If the robot is not moving ($\dot{p} = 0$), it is assumed that the robot will not collide with an obstacle. The proof for obstacle avoidance is explained through the principle of contradiction similar to the proof shown in Chang and Marsden [31].

Let the system collide with the obstacle at t_c with non-zero velocity $\dot{p}(t_c) \neq 0$. The dynamics before the collision during the interval $I = [t_c - \Delta t, t_c^-]$ is considered. The closed loop translational dynamics of the underactuated system can be written as

$$\ddot{p} = g + f = -k_p(p - p_d) - (k_v I_{3\times 3} + G)\dot{p} + z_1 \tag{7.28}$$

Integrating the dynamics for the interval I gives $\dot{p}(t_c^-)$ as

$$\dot{p}(t_c^-) = e^{-k_v \Delta t} R_G(t_c^-, t_c^- - \Delta t)\dot{p}(t_c - \Delta t) + e(\Delta t) \tag{7.29}$$

$$\text{where } e(\Delta t) = \int_{t_c^- - \Delta t}^{t_c^-} e^{-k_v(t_c^- - \tau)} R_G(t_c^-, \tau)(-k_p \Delta p + z_1) d\tau, \tag{7.30}$$

and $R_G(t, \tau)$ is given by solving $\frac{d}{dt} R_G(t, \tau) = -G(x) R_G(t, \tau)$ with $R_G(\tau, \tau) = I$ [36]. The obstacle avoidance matrix $G(x)$ has the form shown in (7.17). The steering axis $e_i(x)$ is assumed to be constant during the small time Δt. For a cylindrical obstacle, the steering axis is chosen to be the major axis of the cylinder, hence it is constant during Δt. For a sphere, this is a valid approximation assuming the displacement vector does not change during the interval I. Under this assumption, the rotation matrix $R_G(t, \tau)$ can be simplified as

$$\frac{d}{dt}R_G(t,\tau) = -k_1(\theta)k_2(d)\hat{e}R_G(t,\tau)$$

$$R_G(t,\tau) = e^{-\left[\int_\tau^t k_1(\theta)k_2(d)dt\right]\hat{e}} = R_e(-\psi(t,\tau)),$$

$$\text{where } \psi(t,\tau) = \int_\tau^t k_1(\theta)k_2(d)dt.$$

The form of $R_e(\psi(t,\tau))$ implies that the obstacle avoidance matrix induces a rotation about the steering axis e, where e is assumed to be constant during Δt and the amount of rotation is given by the angle $\psi(t,\tau)$.

Let the initial value of the Lyapunov function (7.14) be $V_{max} \triangleq V_2(t=0)$. It has been shown in (7.15) that the Lyapunov energy is non-increasing over time. The error vector $e(\Delta t)$ from (7.30) is shown to be $O(\Delta t)$ (i.e the norm of the vector is $O(\Delta t)$) as follows

$$\begin{aligned}
\|e(\Delta t)\| &\leq \int_{t_c-\Delta t}^{t_c^-} \|(-k_p\Delta p + z_1)\|d\tau \\
&\leq \int_{t_c-\Delta t}^{t_c^-} k_p\|\Delta p\| + \|z_1\|d\tau \\
&\leq \sqrt{V_{max}}(\sqrt{2k_p}+1)\Delta t \quad \text{(using (7.10,7.14))}
\end{aligned}$$

Using this result, the velocity at t_c^- from (7.29) can be written as

$$\dot{p}(t_c^-) = e^{-k_v\Delta t}R_e(-\psi(t_c^-, t_c^- - \Delta t))\dot{p}(t_c - \Delta t) + O(\Delta t) \qquad (7.31)$$

Since the Lyapunov energy is non-increasing over time and $\frac{1}{2}m\|\dot{p}\|^2 < V(t) \leq V_{max}$ according to (7.14,7.10), the norm of the velocity of the system is upper bounded as $\|\dot{p}(t)\| < \sqrt{2V_{max}}$. This implies the projected velocity $v(t)$ shown in (7.26, 7.18) should also have the same property, i.e. $\|v(t)\| \leq \sqrt{2V_{max}}$. The derivative of the projected distance vector $d(t)$ shown in (7.20,7.27) is evaluated to be the negative of the projected velocity $\dot{d}(t) = -v(t)$. The projected distance at the time of collision is equal to the radius of the obstacle $\|d(t_c)\| = r$. The projected distance at $t_c - \Delta t$ can be bounded as

$$\|d(t_c - \Delta t)\| \leq \sqrt{2V_{max}}\Delta t + r \tag{7.32}$$

The projected distance is assumed to be continuously getting closer to the obstacle as in

$$\|d(t)\| \leq \|d(t_c - \Delta t), \| \ \ \forall t \in I \tag{7.33}$$

Since the robot is heading towards the obstacle, the absolute value of the heading of the robot from the obstacle will be less than $\pi/2$. Hence, the angular avoidance gain during the interval I can be bounded as $k_1(\theta(t)) \geq e^{-k_{att}}$. The robot is also assumed to be within the detection radius. Hence, the obstacle avoidance gain is simplified as $k_2(d(p(t))) = k_{obs}/(\|d(p(t))\| - r)$. A lower bound on the rotation $\psi(t_c^-, t_c^- - \Delta t)$ is found using above results as

$$\begin{aligned}
\psi(t_c^-, t_c^- - \Delta t) &= \int_{t_c - \Delta t}^{t_c^-} k_1(\theta(t))k_2(d(t))dt \\
&\geq \int_{t_c - \Delta t}^{t_c^-} e^{-k_{att}} \frac{k_{obs}}{\|d(t)\| - r} dt \\
&\geq e^{-k_{att}} \frac{k_{obs}}{\sqrt{2V_{max}}},
\end{aligned}$$

using (7.32-7.33). If k_{obs} is chosen as $k_{obs} \geq \pi e^{k_{att}} \sqrt{2V_{max}}$, the lower bound on the rotation induced by the obstacle avoidance matrix is given by $\psi(t_c^-, t_c^- - \Delta t) \geq \pi$. This bound is not dependent on time left to collision Δt. Using (7.31), the velocity at time of collision is rotated around the obstacle axis by more than 180 degrees from the time $t_c - \Delta t$ when the robot is expected to be heading towards the obstacle. This implies that the robot is not heading towards the obstacle at the time of collision which is a contradiction to our initial assumption that the robot collided with the obstacle at nonzero velocity for which the heading angle needs to be towards the obstacle. Even with a small time to collision, the robot can completely avoid the obstacle with the appropriate gain selection of k_{obs}. In practice, this a very conservative gain and smaller values than this have achieved satisfactory results in terms of collision avoidance.

7.2 Navigation of Unmanned Ground Vehicle (UGV) on 3D Unstructured Terrain using Physics Engine Models

7.2.1 Introduction

This section considers autonomous navigation of unmanned ground vehicles (UGVs) on rough unstructured terrains. We specifically focus on high-frequency physics-based motion generation and control, i.e. computing agile forward motions for the next 0-10 seconds, as opposed to longer-horizon planning. High-fidelity 3D simulation of fast wheeled vehicles on rough terrains has, until recently, been considered too computationally expensive for real-time model predictive control. Rough-terrain mobility is a well-studied topic [82], traditionally addressed by compressing the sensed terrain into a planar traversability map as opposed to a high-fidelity 3D deformable mesh capable of simulating tire-soil interaction. The traversability approach enables efficient planning through simplified vehicle models [79, 213] even at high speeds [224]. Typically this approach is integrated with global long-horizon 2D map planning [130]. Recent advances in efficient physics- based simulation could now enable real-time 3d simulation-based control [96] for safely traversing as opposed to simply avoiding unstructured terrain. While high-resolution particulate terrain models [217, 139] might still not be fast enough for real-time optimal control, deformable mesh models with adjustable stiffness, damping, and slip parameters are available and support faster than real-time simulation. Current simulation tools such as the Bullet physics engine [26] offer such functionalities and, coupled with parameter identification and predictive control, could provide significant advances over the planar surface assumption to enable safe off-road traversal and collision avoidance.

7.2.2 UGV Model

The model used in this work is based on a ray-cast model using Bullet physics engine [26].This model has been shown [133, 149] to replicate the actual car model closely when initialized with appropriate model parameters. The physics can generate trajectories efficiently for example a trajectory 10 seconds long running at 100 Hz can be produced in 10 milliseconds.

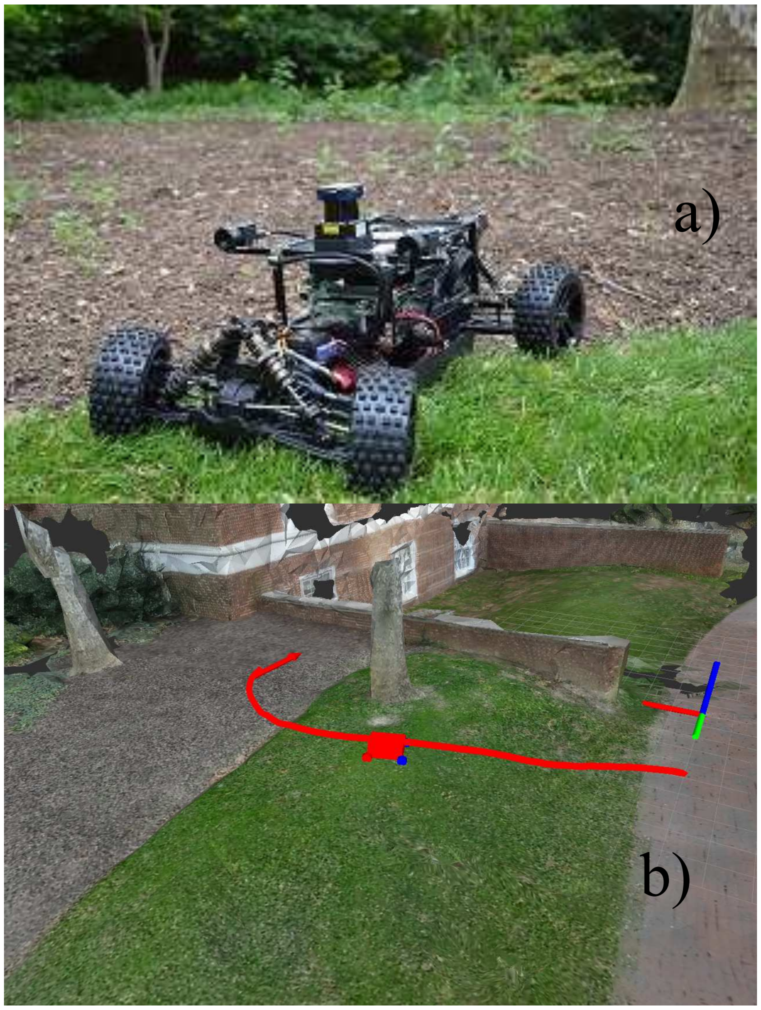


Figure 7.4: a) The JHU 1/5-scale model Unmanned Ground Vehicle (UGV) b) an optimized path to a goal location using receding horizon control (RHC) based on Bullet physics simulation using a terrain from dense visual 3D reconstruction.

7.2.3 Optimal Control Formulation

The goal of the optimization problem is to plan an optimal trajectory for UGV model to reach a desired target state while avoiding obstacles and minimizing control effort in the process. This is a challenging problem, since we do not have an analytical model for the rccar and the terrain is unstructured. The RHC optimization problem is stated as the minimization

$$\begin{aligned} &\min_{\xi_{0:n-1}} l_n(x_n) + \sum_{i=0}^{n-1} l_i(x_i, \xi_i), \quad \text{subject to:} \\ &\xi_i \in \xi, \qquad u_i = \psi(\xi_i, x_i), \quad x_{i+1} = \text{sim}(x_i, u_i), \end{aligned} \tag{7.34}$$

where $\xi \subset \mathbb{R}^2$ denotes the admissible control set, x_i are the discrete states along the trajectory, u_i are the physics engine control inputs, and *sim* is the physics engine step function. We employ a linear quadratic cost that takes into account control cost, state cost and terminal cost to drive the trajectory towards the desired goal x_f while penalizing control effort. the proposed formulation can handle both following reference trajectories $\{x_{r0}, x_{r1}, \ldots\}$ with associated controls $\{\xi_{r0}, \xi_{r1}, \ldots\}$ as well as reaching a single desired goal state x_f: When the UGV collides with obstacles, it automatically fails to reach the goal state. Hence, a simple cost formulation that minimizes the distance to the goal state already accounts for obstacle avoidance, slip, and roll-over assuming the dynamic model is accurate. This control formulation is still prone to local minima around untraversable obstacles such as walls, as well as highly irregular terrain. We address this challenge using a stochastic sampling-based optimization methods such as Cross Entropy (CE) [103] based sampling and Sampling based Differential Dynamic Programming (SDDP) [64,

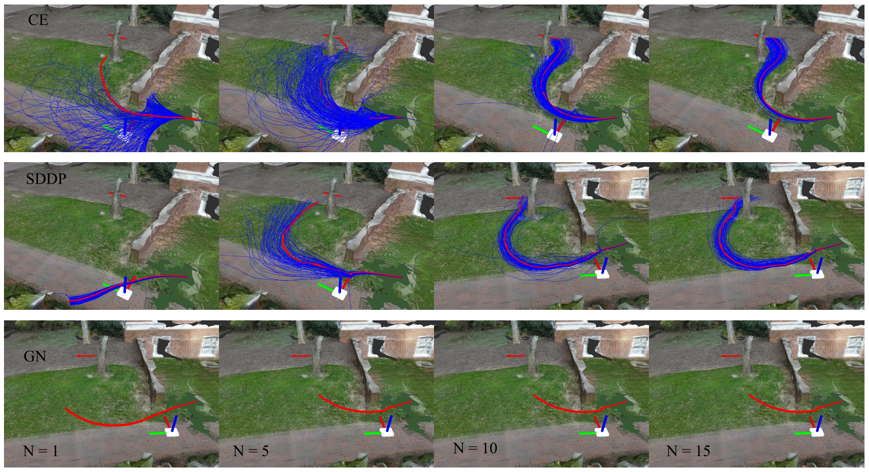


Figure 7.5: Tile of Pictures showing optimal trajectory for different optimization algorithms at N = 15 iterations. These figures show that CE outperforms SDDP and Gauss Newton (GN) in finding nearly globally optimal trajectory.

146].

7.2.4 Results

The Figure 7.5 shows the result of applying the three different trajectory optimization algorithms to the UGV model on a model terrain. The goal of the optimization is to navigate the car on the terrain to a goal location while avoiding the tree and wall in the path. The NMPC optimization is able to achieve the desired goal even without an explicit obstacle cost encoded into the trajectory optimization. Further, we notice that sampling based methods such as CE and SDDP outperform a pure gradient descent method.

7.2.5 Simulation Results and Empirical Analysis

In this section, the results of applying two stochastic sampling methods to the optimization problem described in section 7.2.3 are shown. One of the limitations of the local optimization methods is that, the cost of the final optimal trajectory depends on availability of a good initial guess. Figure 7.6 shows the average optimal trajectory cost and its variance with respect to the random initializations(random $\xi_{0:n-1}$) for different optimization algorithms. It has been observed that the sampling based methods obtain a much lower average and variance optimal cost compared to Gauss Newton (GN) optimization method which is a local optimization method. The results of applying the

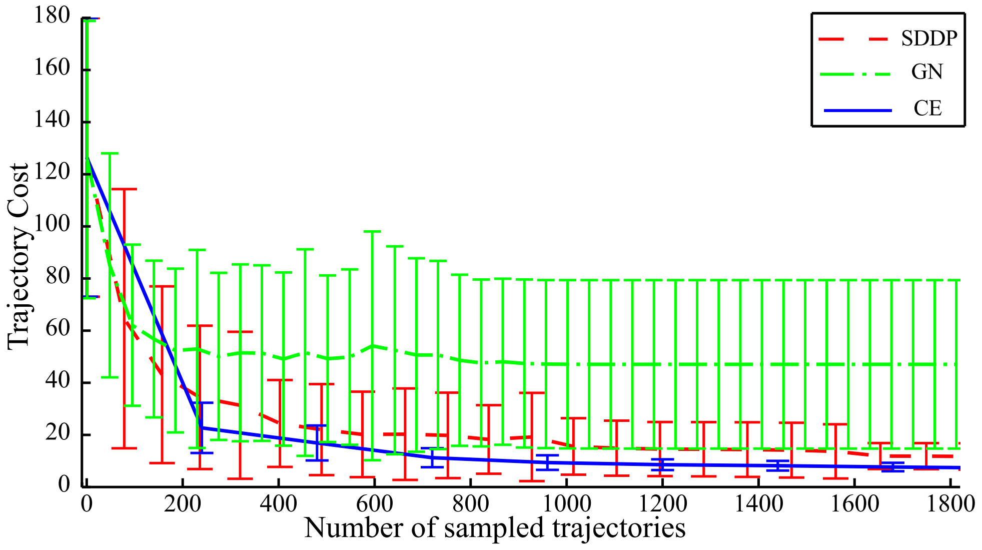


Figure 7.6: Plots showing mean and Variance of the trajectory cost for various initial control guesses for the same optimization problem

optimization algorithms to an example scenario is shown in Figure 7.5. It can be seen that both the stochastic sampling methods find feasible trajectories whereas GN method fails to find a feasible trajectory to the target state.

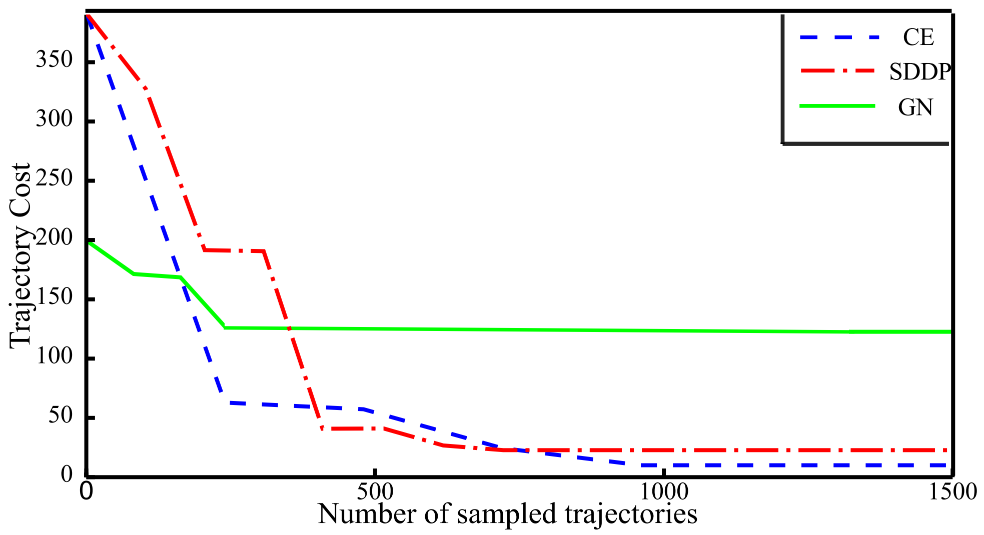


Figure 7.7: Trajectory cost for different algorithms as a function of number of samples. GN method gets stuck in local minimum whereas SDDP and CE are able to find nearly optimal trajectories.

7.2.6 Conclusions

In this section, an optimal control formulation for Receding Horizon Control of UGV on unstructured terrains has been presented. The superiority of stochastic sampling methods to local optimization methods has been shown through simulations.

Chapter 8

Conclusion

In summary, we described different components of a robust control scheme namely: system identification, trajectory optimization, and uncertainty propagation. We first showed that using NMPC we can treat the aerial manipulation system as a single dynamic system. By planning desired trajectories using NMPC, we minimized the time taken to grasp an object as compared to planning using a kinematic trajectory. Next, we introduced adaptive NMPC, where we learned the dynamics of the robot online and performed obstacle avoidance using the learned model. We used the unscented transform to propagate the parametric uncertainty in the model to the state space and use this as a buffer for avoiding obstacles with high probability. We showed through experiments that the quadrotor is indeed able to avoid obstacles at speeds up to 4 m/s. After that, we used recurrent neural networks to model the dynamics of an aerial manipulator and a passenger vehicle. We modified a vanilla RNN to accept prior information about the model in the form of feedforward input. We showed that using the feedforward input reduced the size of network with a slight increase in the number of model parameters. We further combined the learned model with NMPC to control the robot. We showed that using the learned model performed comparably to first-principles models but without the necessity of extensive knowledge about the system dynamics.

Later on, we introduced a numerical method to compute invariant sets for propagating model uncertainty for a general nonlinear system paired with a nonlinear controller. We showed that combining the numerical method with MPC can be used to compute approximately safe trajectories for obstacle avoidance. Unlike the adaptive NMPC introduced before, this approach propagates the uncertainty in closed loop and thus the ellipsoids shrink to some constant value over time when using a stable controller.

Finally, we also described a state-machine framework that combines all the designed controllers in a safe manner to achieve high-level tasks. We showed that the state-machine framework can perform recovery actions based on controller/hardware failures.

8.1 Future Work

Future work should expand on the three components of the robust control scheme:

System Identification: Current work described using MLE to identify parametric models for dynamic systems online. These models are limited by our knowledge of the robotic system. We introduced neural network models to overcome this limitation, but currently, we could only learn neural network models offline. Future work should learn the neural network models online to account for any time-varying components. Using neural networks to detect failures/outliers in sensors will also make system identification robust.

Uncertainty Propagation: We described propagating uncertainty from model parameters to state space using an unscented transform. Later on, we propagated uncertainty in closed-loop by computing the disturbance invariant set. The invariant set computation requires the knowledge of the bound on external disturbances. In the future, the disturbance bound can be learned from samples collected online allowing us to update invariant funnels based on learning dynamics online. Computing the invariant funnels in real-time for high dimensional systems is another area of future research.

Trajectory optimization: Currently, we used local trajectory optimization methods to solve boundary value problems such as reaching a terminal state and trajectory following tasks while avoiding obstacles. These methods should be combined in the future with global planning methods to perform long distance missions. Reasoning about uncertainty when performing global planning is another avenue to be researched.

Future work should also focus on applying robust control to other complete applications in aerial manipulation such package transportation and delivery, picking produce in agriculture, and high-altitude servicing.

References

[1] *AEROARMS*. https://aeroarms-project.eu/. 2017.

[2] *AeroWorks*. http://www.aeroworks2020.eu/. 2017.

[3] *Airobots*. http://airobots.ing.unibo.it. 2015.

[4] A. C. Aitken. “IV.âĂŤOn Least Squares and Linear Combination of Observations”. In: *Proceedings of the Royal Society of Edinburgh* 55 (1936), 42âĂŞ48. DOI: 10.1017/S0370164600014346.

[5] Saif A Al-Hiddabi. “Quadrotor control using feedback linearization with dynamic extension”. In: *Mechatronics and its Applications, 2009. ISMA'09. 6th International Symposium on*. IEEE. 2009, pp. 1–3.

[6] Ross Allen and Marco Pavone. “A Real-Time Framework for Kinodynamic Planning with Application to Quadrotor Obstacle Avoidance”. In: *AIAA Conf. on Guidance, Navigation and Control, San Diego, CA*. 2016.

[7] F. Allgöwer et al. “Nonlinear Predictive Control and Moving Horizon Estimation — An Introductory Overview”. In: *Advances in Control*. Ed. by Paul M. Frank. London: Springer London, 1999, pp. 391–449. ISBN: 978-1-4471-0853-5.

[8] Erdinç Altuğ, James P Ostrowski, and Robert Mahony. “Control of a quadrotor helicopter using visual feedback”. In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 1. IEEE. 2002, pp. 72–77.

[9] *Alvar*. http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html. 2017.

[10] Badawy Aly et al. “Modeling and Analysis of an Electric Power Steering System”. In: *SAE Technical Paper*. Society of Automotive Engineers(SAE) International, 1999. DOI: 10.4271/1999-01-0399. URL: http://dx.doi.org/10.4271/1999-01-0399.

[11] Amazon. *Prime Air*. https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011. 2017.

[12] Joel A E Andersson et al. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* (In Press, 2018).

[13] *ARCAS*. http://www.arcas-project.eu. 2015.

[14] A. Aswani et al. "Reducing Transient and Steady State Electricity Consumption in HVAC Using Learning-Based Model-Predictive Control". In: *Proceedings of the IEEE* 100.1 (2012), pp. 240–253. ISSN: 0018-9219. DOI: 10.1109/JPROC.2011.2161242.

[15] Anil Aswani et al. "Provably safe and robust learning-based model predictive control". In: *Automatica* 49.5 (2013), pp. 1216–1226.

[16] A. J. Barry et al. "Flying between obstacles with an autonomous knife-edge maneuver". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 2559–2559. DOI: 10.1109/ICRA.2014.6907217.

[17] Carmine Dario Bellicoso et al. "Design, modeling and control of a 5-DoF light-weight robot arm for aerial manipulation". In: *Control and Automation (MED), 2015 23th Mediterranean Conference on*. IEEE. 2015, pp. 853–858.

[18] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.

[19] D. P. Bertsekas. *Nonlinear Programming, 2nd ed.* Belmont, MA: Athena Scientific, 2003.

[20] Peter J Bickel and Kjell A Doksum. *Mathematical statistics: basic ideas and selected topics, volume I*. Vol. 117. CRC Press, 2015.

[21] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[22] *Boost Meta State Machine MSM Library*. 2017.

[23] Samir Bouabdallah and Roland Siegwart. "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 2247–2252.

[24] P Bouffard, A Aswani, and C Tomlin. "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results". In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 279–284. ISBN: 1050-4729.

[25] A. Budiyanto et al. "UAV obstacle avoidance using potential field under dynamic environment". In: *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*. 2015, pp. 187–192.

[26] *Bullet Physics Library*.
urlhttp://bulletphysics.org/.

[27] Francesco Bullo and Andrew Lewis. *Geometric Control of Mechanical Systems*. Springer, 2004.

[28] Eric Caillibot, Cordell Grant, and Daniel Kekez. "Formation flying demonstration missions enabled by CanX nanosatellite technology". In: (2005).

[29] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[30] Adrian Carrio et al. "A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles". In: *Journal of Sensors* 2017 (2017).

[31] Dong Eui Chang and Jerrold E Marsden. "Gyroscopic forces and collision avoidance with convex obstacles". In: *New trends in nonlinear dynamics and control and their applications*. Springer, 2003, pp. 145–159.

[32] Dong Eui Chang et al. "Collision avoidance for multiple agent systems". In: (2003).

[33] Kai Chang et al. "Obstacle avoidance and active disturbance rejection control for a quadrotor". In: *Neurocomputing* (2016).

[34] Abraham Charnes and William W Cooper. "Chance-constrained programming". In: *Management science* 6.1 (1959), pp. 73–79.

[35] Chenyi Chen et al. "Deepdriving: Learning affordance for direct perception in autonomous driving". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2722–2730.

[36] Chi-Tsong Chen. *Linear system theory and design*. Oxford University Press, Inc., 1995.

[37] Hong Chen, Carsten W Scherer, and F Allgower. "A game theoretic approach to nonlinear robust receding horizon control of constrained systems". In: *American Control Conference*. Vol. 5. IEEE. 1997, pp. 3073–3077.

[38] Changhyun Choi and Henrik I Christensen. "Real-time 3D model-based tracking using edge and keypoint features for robotic manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 4048–4055.

[39] Yin-Lam Chow and Marco Pavone. "A framework for time-consistent, risk-averse model predictive control: Theory and algorithms". In: *American Control Conference (ACC), 2014*. IEEE. 2014, pp. 4204–4211.

[40] Benoît Colson, Patrice Marcotte, and Gilles Savard. "An overview of bilevel optimization". In: *Annals of operations research* 153.1 (2007), pp. 235–256.

[41] Massimo Conti and Claudio Turchetti. "Approximation of dynamical systems by continuous-time recurrent approximate identity neural networks". In: *Neural, Parallel and Scientific Computations* 2.3 (1994), pp. 299–320.

[42] Rita Cunha, David Cabecinhas, and Carlos Silvestre. "Nonlinear trajectory tracking control of a quadrotor vehicle". In: *2009 European Control Conference (ECC)*. IEEE. 2009, pp. 2763–2768.

[43] Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. "Feedback control of a nonholonomic car-like robot". In: *Robot motion planning and control*. Springer, 1998, pp. 171–253.

[44] Morris H DeGroot. *Optimal statistical decisions*. Vol. 82. John Wiley & Sons, 2005.

[45] Morris H DeGroot and Mark J Schervish. *Probability and statistics*. Pearson Education, 2012.

[46] Vishnu R Desaraju, Alexander E Spitzer, and Nathan Michael. "Experience-driven Predictive Control with Robust Constraint Satisfaction under Time-Varying State Uncertainty". In: *Robotics: Science and Systems Conference (RSS), July*. 2017.

[47] *DJI A3 autopilot*. 2018. URL: https://www.dji.com/a3.

[48] *DJI Guidance Pro Suite*. https://www.dji.com/ground-station-pro. 2017.

[49] *DJI Guidance Sensor*. http://www.dji.com/product/guidance. 2015.

[50] *DJI Matrice*. http://www.dji.com/. 2015.

[51] Zachary T Dydek, Anuradha M Annaswamy, and Eugene Lavretsky. "Adaptive control of quadrotor UAVs: A design trade study with flight evaluations". In: *IEEE Transactions on control systems technology* 21.4 (2013), pp. 1400–1406.

[52] K T R Van Ende et al. "Practicability study on the suitability of artificial, neural networks for the approximation of unknown steering torques". In: *Vehicle System Dynamics* 54.10 (2016), pp. 1362–1383. DOI: 10.1080/00423114.2016.1202987. URL: http://dx.doi.org/10.1080/00423114.2016.1202987.

[53] B. Erginer and E. Altug. "Modeling and PD Control of a Quadrotor VTOL Vehicle". In: *2007 IEEE Intelligent Vehicles Symposium*. 2007, pp. 894–899.

[54] Morteza Farrokhsiar and Homayoun Najjaran. "Unscented model predictive control of chance constrained nonlinear systems". In: *Advanced Robotics* 28.4 (2014), pp. 257–267.

[55] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2008.

[56] Jonathan Fink et al. "Planning and control for cooperative manipulation and transportation with aerial robots". In: *The International Journal of Robotics Research* 30.3 (2011), pp. 324–334.

[57] D. Fusata, G. Guglieri, and R. Celi. "Flight Dynamics of an Articulated Rotor Helicopter with an External Slung Load". In: *Journal of the American Helicopter Society* 46.1 (2001), pp. 3–14.

[58] Brendan Galea et al. "Stippling with aerial robots". In: *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*. Eurographics Association. 2016, pp. 125–134.

[59] Yiqi Gao et al. "A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles". In: *Vehicle System Dynamics* 52.6 (2014), pp. 802–823.

[60] G. Garimella et al. "Neural network modeling for steering control of an autonomous vehicle". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 2609–2615. DOI: 10.1109/IROS.2017.8206084.

[61] Abel Gawel et al. “Aerial picking and delivery of magnetic objects with mavs”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5746–5752.

[62] Qingbo Geng, Huan Shuai, and Qiong Hu. “Obstacle avoidance approaches for quadrotor UAV based on backstepping technique”. In: *Control and Decision Conference (CCDC), 2013 25th Chinese*. IEEE. 2013, pp. 3613–3617.

[63] Marcel van Gerven and Sander Bohte. *Artificial neural networks as models of neural information processing*. Frontiers Media SA, 2018.

[64] Vaibhav Ghadiok, Jeremy Goldin, and Wei Ren. “Autonomous indoor aerial gripping using a quadrotor”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 4645–4651.

[65] Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. “Geometric nonlinear PID control of a quadrotor UAV on SE (3)”. In: *preprint arXiv:1304.6765* (2013).

[66] Geoff Gordon and Ryan Tibshirani. “Coordinate descent”. In: *Optimization* 10.725/36 (2015), p. 725.

[67] David Gunning. “Explainable artificial intelligence (xai)”. In: *Defense Advanced Research Projects Agency (DARPA), nd Web* (2017).

[68] Shweta Gupte, Paul Infant Teenu Mohandas, and James M Conrad. “A survey of quadrotor Unmanned Aerial Vehicles”. In: *2012 Proceedings of IEEE Southeastcon*. IEEE. 2012, pp. 1–6. DOI: 10.1109/SECon.2012.6196930.

[69] A Haddoun et al. “Modeling, Analysis, and Neural Network Control of an EV Electrical Differential”. In: *IEEE Transactions on Industrial Electronics* 55.6 (2008), pp. 2286–2294. DOI: 10.1109/TIE.2008.918392.

[70] Erik Hallstrom. 2016. URL: https://medium.com/@erikhallstrm/hello-world-rnn-83cd7105b767.

[71] PC Hansen, V Pereyra, and G Scherer. “Nonlinear least squares problems”. In: (2004).

[72] Monson H Hayes. *Statistical digital signal processing and modeling*. John Wiley & Sons, 2009.

[73] M. Hehn and R. D’Andrea. “A flying inverted pendulum”. In: *IEEE International Conference on Robotics and Automation (ICRA), 2011*. 2011, pp. 763 –770. DOI: 10.1109/ICRA.2011.5980244.

[74] G. Heredia et al. “Control of a multirotor outdoor aerial manipulator”. In: *International Conference on Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ*. 2014, pp. 3417–3422. DOI: 10.1109/IROS.2014.6943038.

[75] G. Heredia et al. “Control of a multirotor outdoor aerial manipulator”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 3417–3422. DOI: 10.1109/IROS.2014.6943038.

[76] Jonas Heylen et al. “From Pixels to Actions: Learning to Drive a Car with Deep Neural Networks”. In: *IEEE Winter Conference on Applications in Computer Vision*. 2018.

[77] Desmond J. Higham. “An algorithmic introduction to numerical simulation of stochastic differential equations”. In: *SIAM Review* 43 (2001), pp. 525–546.

[78] Robert V Hogg and Allen T Craig. *Introduction to mathematical statistics.(5”” edition)*. Upper Saddle River, New Jersey: Prentice Hall, 1995.

[79] ThomasM. Howard, ColinJ. Green, and Alonzo Kelly. “Receding Horizon Model-Predictive Control for Mobile Robot Navigation of Intricate Paths”. English. In: *Field and Service Robotics*. Ed. by Andrew Howard, Karl Iagnemma, and Alonzo Kelly. Vol. 62. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2010, pp. 69–78. ISBN: 978-3-642-13407-4. DOI: 10.1007/978-3-642-13408-1_7. URL: http://dx.doi.org/10.1007/978-3-642-13408-1_7.

[80] Albert S. Huang et al. “Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera”. In: *Robotics Research : The 15th International Symposium ISRR*. Ed. by Henrik I. Christensen and Oussama Khatib. Cham: Springer International Publishing, 2017, pp. 235–252. ISBN: 978-3-319-29363-9. DOI: 10.1007/978-3-319-29363-9_14. URL: https://doi.org/10.1007/978-3-319-29363-9_14.

[81] C. Huerzeler et al. “Applying aerial robotics for inspections of power and petrochemical facilities”. In: *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*. 2012, pp. 167–172. DOI: 10.1109/CARPI.2012.6473371.

[82] Karl Iagnemma and Steven Dubowsky. “Rough Terrain Control”. In: *Mobile Robots in Rough Terrain*. Springer Berlin Heidelberg, 2004, pp. 81–96.

[83] Ian Lenz AND Ross Knepper AND Ashutosh Saxena. “Deepmpc: Learning deep latent features for model predictive control”. In: *Robotics: Science and Systems*. Rome, Italy, 2015. DOI: 10.15607/RSS.2015.XI.012.

[84] A. Ichikawa et al. “UAV with manipulator for bridge inspection; Hammering system for mounting to UAV”. In: *2017 IEEE/SICE International Symposium on System Integration (SII)*. 2017, pp. 775–780. DOI: 10.1109/SII.2017.8279316.

[85] David H. Jacobson and David Q. Mayne. *Differential dynamic programming*. Modern analytic and computational methods in science and mathematics. New York: Elsevier, 1970. ISBN: 0-444-00070-4. URL: http://opac.inria.fr/record=b1078358.

[86] A. K. Jain, Jianchang Mao, and K. M. Mohiuddin. “Artificial neural networks: a tutorial”. In: *Computer* 29.3 (1996), pp. 31–44. ISSN: 0018-9162. DOI: 10.1109/2.485891.

[87] Abhinandan Jain. *Robot and Multibody Dynamics: Analysis and Algorithms*. Springer, 2011.

[88] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1999. ISBN: 0849371813.

[89] SH Jeong and S Jung. “Bilateral Teleoperation Control of a Quadrotor System with a Haptic Device: Experimental Studies”. In: (2014).

[90] A. E. Jimenez-Cano et al. “Control of an aerial robot with multi-link arm for assembly tasks”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 4916–4921. DOI: 10.1109/ICRA.2013.6631279.

[91] A.E. Jimenez-Cano et al. “Control of an aerial robot with multi-link arm for assembly tasks”. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. 2013, pp. 4916–4921. DOI: 10.1109/ICRA.2013.6631279.

[92] AE Jimenez-Cano et al. “Aerial manipulator for structure inspection by contact from the underside”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 1879–1884.

[93] Eric N Johnson and Suresh K Kannan. "Adaptive Trajectory Control for Autonomous Helicopters". In: *Journal of Guidance, Control, and Dynamics* 28.3 (2005), pp. 524–538. ISSN: 0731-5090. DOI: 10.2514/1.6271. URL: http://dx.doi.org/10.2514/1.6271.

[94] Rudolf Emil Kalman et al. "Contributions to the theory of optimal control". In: *Bol. Soc. Mat. Mexicana* 5.2 (1960), pp. 102–119.

[95] Rudolph Emil Kalman. "A new approach to linear filtering and prediction problems". In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[96] Nima Keivan and Gabe Sibley. "Realtime Simulation-in-the-loop Control for Agile Ground Vehicles". In: *TAROS*. 2014.

[97] Klaas Kelchtermans and Tinne Tuytelaars. "How hard is it to cross the room?–Training (Recurrent) Neural Networks to steer a UAV". In: *arXiv preprint arXiv:1702.07600* (2017).

[98] Hassan K Khalil. "Noninear systems". In: *Prentice-Hall, New Jersey* 2.5 (1996), pp. 5–1.

[99] Hassan K Khalil and JW Grizzle. *Nonlinear systems*. Vol. 3. Prentice hall New Jersey, 1996.

[100] Dongbin Kim and Paul Y Oh. "Lab automation drones for mobile manipulation in high throughput systems". In: *IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2018, pp. 1–5.

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

[102] Paisan Kittisupakorn et al. "Neural network based model predictive control for a steel pickling process". In: *Journal of Process Control* 19.4 (2009), pp. 579–590. DOI: 10.1016/j.jprocont.2008.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0959152408001388.

[103] M. Kobilarov. "Cross-entropy Motion Planning". In: *International Journal of Robotics Research* 31.7 (2012), pp. 855–871.

[104] M. Kobilarov. "Discrete optimal control on Lie groups and applications to robotic vehicles". In: *IEEE International Conference on Robotics and Automation*. 2014, pp. 5523–5529.

[105] M. Kobilarov. “Nonlinear Trajectory Control of Multi-body Aerial Manipulators”. In: *Journal of Intelligent & Robotic Systems* 73.1-4 (2014), pp. 679–692.

[106] M. Kobilarov and J.E. Marsden. “Discrete Geometric Optimal Control on Lie Groups”. In: *IEEE Transactions on Robotics,* 27.4 (2011), pp. 641–655. ISSN: 1552-3098. DOI: 10.1109/TRO.2011.2139130.

[107] Marin Kobilarov. “Trajectory tracking of a class of underactuated systems with external disturbances”. In: *American Control Conference*. 2013, pp. 1044–1049.

[108] Marin Kobilarov. “Trajectory tracking of a class of underactuated systems with external disturbances”. In: *American Control Conference (ACC), 2013*. IEEE. 2013, pp. 1044–1049.

[109] Marin Kobilarov. “Nonlinear trajectory control of multi-body aerial manipulators”. In: *Journal of Intelligent & Robotic Systems* 73.1-4 (2014), pp. 679–692.

[110] Marin Kobilarov and Sergio Pellegrino. “Trajectory planning for CubeSat short-time-scale proximity operations”. In: *Journal of Guidance, Control, and Dynamics* 37.2 (2014), pp. 566–579.

[111] William Koehrsen. 2017. URL: https://medium.com/@williamkoehrsen/deep-neural-network-classifier-32c12ff46b6c.

[112] Ilya Kolmanovsky and Elmer G Gilbert. “Theory and computation of disturbance invariant sets for discrete-time linear systems”. In: *Mathematical problems in engineering* 4.4 (1998), pp. 317–367.

[113] K. Kondak et al. “Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 2107–2112.

[114] K. Kondak et al. “Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 2107–2112.

[115] Konstantin Kondak et al. “Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2107–2112.

[116] Konstantin Kondak et al. “Unmanned Aerial Systems Physically Interacting with the Environment: Load Transportation, Deployment, and Aerial Manipulation”. In: *Handbook of Unmanned Aerial Vehicles*. Ed. by Kimon P. Valavanis and George J. Vachtsevanos. Dordrecht: Springer Netherlands, 2015, pp. 2755–2785. DOI: 10.1007/978-90-481-9707-1_77. URL: https://doi.org/10.1007/978-90-481-9707-1_77.

[117] C. Korpela et al. “Flight stability in aerial redundant manipulators”. In: *IEEE International Conference on Robotics and Automation (ICRA), 2012*. 2012, pp. 3529 –3530. DOI: 10.1109/ICRA.2012.6224925.

[118] Christopher Korpela et al. “Towards the realization of mobile manipulating unmanned aerial vehicles (MM-UAV): Peg-in-hole insertion tasks”. In: *International Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE, 2013, pp. 1–6.

[119] C.M. Korpela, T.W. Danko, and P.Y. Oh. “Designing a system for mobile manipulation from an Unmanned Aerial Vehicle”. In: *Technologies for Practical Robot Applications (TePRA), 2011 IEEE Conference on*. 2011, pp. 109 –114. DOI: 10.1109/TEPRA.2011.5753491.

[120] Mayuresh V Kothare, Venkataramanan Balakrishnan, and Manfred Morari. “Robust constrained model predictive control using linear matrix inequalities”. In: *Automatica* 32.10 (1996), pp. 1361–1379.

[121] Krisada Kritayakirana and J Christian Gerdes. “Autonomous vehicle control at the limits of handling”. PhD thesis. Stanford University, 2012, p. 215.

[122] S Kumarawadu and T T Lee. “Neuroadaptive Combined Lateral and Longitudinal Control of Highway Vehicles Using RBF Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* 7.4 (2006), pp. 500–512. DOI: 10.1109/TITS.2006.883113.

[123] Daewon Lee, H Jin Kim, and Shankar Sastry. “Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter”. In: *International Journal of control, Automation and systems* 7.3 (2009), pp. 419–428.

[124] Hyeonbeom Lee et al. “An Integrated Framework for Cooperative Aerial Manipulators in Unknown Environments”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2307–2314.

[125] Taeyoung Lee, Melvin Leoky, and N Harris McClamroch. “Geometric tracking control of a quadrotor UAV on SE (3)”. In: *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE. 2010, pp. 5420–5425.

[126] C Lehnert and G Wyeth. “Locally Weighted Learning Model Predictive Control for nonlinear and time varying dynamics”. In: *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 2619–2625. ISBN: 1050-4729.

[127] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. “DeepMPC: Learning Deep Latent Features for Model Predictive Control.” In: *Robotics: Science and Systems*. 2015.

[128] I J Leontaritis and Stephen A Billings. “Input-output parametric models for non-linear systems Part I: deterministic non-linear systems”. In: *International journal of control* 41.2 (1985), pp. 303–328.

[129] Pu Li, Harvey Arellano-Garcia, and Günter Wozny. “Chance constrained programming approach to process optimization under uncertainty”. In: *Computers & chemical engineering* 32.1 (2008), pp. 25–45.

[130] Maxim Likhachev et al. “Anytime search in dynamic graphs”. In: *Artif. Intell.* 172.14 (2008), pp. 1613–1643. ISSN: 0004-3702. DOI: 10.1016/j.artint.2007.11.009. URL: http://dx.doi.org/10.1016/j.artint.2007.11.009.

[131] Quentin J. Lindsey, Daniel Mellinger, and Vijay Kumar. “Construction of Cubic Structures with Quadrotor Teams”. In: *Robotics: Science and Systems* (2011).

[132] Vincenzo Lippiello and Fabio Ruggiero. “Exploiting redundancy in cartesian impedance control of uavs equipped with a robotic arm”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012*. IEEE. 2012, pp. 3768–3773.

[133] Vincenzo Lippiello et al. “Hybrid visual servoing with hierarchical task composition for aerial manipulation”. In: *IEEE Robotics and Automation Letters* 1.1 (2016), pp. 259–266.

[134] Yuyi Liu et al. “A robust nonlinear controller for nontrivial quadrotor maneuvers: Approach and verification”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 5410–5416.

[135] C H Lu and C C Tsai. "Adaptive Predictive Control With Recurrent Neural Network for Industrial Processes: An Application to Temperature Control of a Variable-Frequency Oil-Cooling Machine". In: *IEEE Transactions on Industrial Electronics* 55.3 (2008), pp. 1366–1375. DOI: 10.1109/TIE.2007.896492.

[136] D. Lunni et al. "Nonlinear model predictive control for aerial manipulation". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017, pp. 87–93.

[137] Raymond R Ma, Lael U Odhner, and Aaron M Dollar. "A modular, open-source 3d printed underactuated hand". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 2737–2743.

[138] BG Maciel-Pearson and TP Breckon. "An Optimised Deep Neural Network Approach for Forest Trail Navigation for UAV Operation within the Forest Canopy". In: (2017).

[139] J. Madsen, A. Seidl, and D. Negrut. "Compaction-Based Deformable Terrain Model as an Interface for Real-Time Vehicle Dynamics Simulations". In: *SAE Technical Paper*. 2013.

[140] Robert Mahony and Tarek Hamel. "Robust trajectory tracking for a scale model autonomous helicopter". In: *International Journal of Robust and Nonlinear Control* 14.12 (2004), pp. 1035–1059.

[141] Anirudha Majumdar and Russ Tedrake. "Funnel Libraries for Real-Time Robust Feedback Motion Planning". In: *arXiv preprint arXiv:1601.04037* (2016).

[142] Anirudha Majumdar and Russ Tedrake. "Funnel libraries for real-time robust feedback motion planning". In: *The International Journal of Robotics Research* 36.8 (2017), pp. 947–982.

[143] Ian R Manchester and Jean-Jacques E Slotine. "Control contraction metrics and universal stabilizability". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 8223–8228.

[144] Ian R Manchester and Jean-Jacques E Slotine. "Output-feedback control of nonlinear systems using control contraction metrics and convex optimization". In: *Australian Control Conference (AUCC)*. IEEE. 2014, pp. 215–220.

[145] Zachary Manchester and Scott Kuindersma. “DIRTREL: Robust Trajectory Optimization with Ellipsoidal Disturbances and LQR Feedback”. In: (2017).

[146] L. Marconi et al. “Aerial service robots: An overview of the AIRobots activity”. In: *Applied Robotics for the Power Industry (CARPI), 2012 2nd International Conference on*. IEEE, 2012, pp. 76–77.

[147] David Q Mayne, María M Seron, and SV Raković. “Robust model predictive control of constrained linear systems with bounded disturbances”. In: *Automatica* 41.2 (2005), pp. 219–224.

[148] I. Maza et al. “Multi-UAV Cooperation and Control for Load Transportation and Deployment”. English. In: *Journal of Intelligent and Robotic Systems* 57.1-4 (2010), pp. 417–449. ISSN: 0921-0296. DOI: 10.1007/s10846-009-9352-8. URL: http://dx.doi.org/10.1007/s10846-009-9352-8.

[149] Rafik Mebarki and Vincenzo Lippiello. “ImageâĂŘBased Control for Aerial Manipulation”. In: *Asian Journal of Control* 16.3 (2014), pp. 646–656.

[150] Lorenz Meier. *Pixhawk AutoPilot - PX4 Autopilot Platform*. 2014. URL: http://pixhawk.org/modules/pixhawk.

[151] Daniel Mellinger et al. “Design, modeling, estimation and control for aerial grasping and manipulation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 2668–2673.

[152] A.Y. Mersha, S. Stramigioli, and R. Carloni. “Bilateral teleoperation of underactuated unmanned aerial vehicles: The virtual slave concept”. In: *IEEE International Conference on Robotics and Automation(ICRA)*. 2012, pp. 4614–4620. DOI: 10.1109/ICRA.2012.6224711.

[153] Nathan Michael, Jonathan Fink, and Vijay Kumar. “Cooperative manipulation and transportation with aerial robots”. In: *Autonomous Robots* 30.1 (2011), pp. 73–86. ISSN: 1573-7527. DOI: 10.1007/s10514-010-9205-0. URL: https://doi.org/10.1007/s10514-010-9205-0.

[154] William F Milliken and Douglas L Milliken. *Race car vehicle dynamics*. Vol. 400. Society of Automotive Engineers Warrendale, 1995.

[155] Haibo Min, Fuchun Sun, and Feng Niu. "Decentralized UAV formation tracking flight control using gyroscopic force". In: *Computational Intelligence for Measurement Systems and Applications, 2009. CIMSA '09. IEEE International Conference on*. 2009, pp. 91–96. DOI: 10.1109/CIMSA.2009.5069925.

[156] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *Deep Learning Workshop at Neural Information Processing Systems NIPS* (2013). URL: https://arxiv.org/abs/1312.5602.

[157] Javier Molina and Shinichi Hirai. "Pruning tree-branches close to electrical power lines using a skew-gripper and a multirotor helicopter". In: *Advanced Intelligent Mechatronics (AIM), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1123–1128.

[158] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC, 1994.

[159] M. Neunert et al. "Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1398–1404. DOI: 10.1109/ICRA.2016.7487274.

[160] Hai-Nguyen Nguyen et al. "A novel robotic platform for aerial manipulation using quadrotors as rotating thrust generators". In: *IEEE Transactions on Robotics* 34.2 (2018), pp. 353–369.

[161] C Nicol, C J B Macnab, and A Ramirez-Serrano. "Robust adaptive control of a quadrotor helicopter". In: *Mechatronics* 21.6 (2011), pp. 927–938. ISSN: 0957-4158. DOI: http://dx.doi.org/10.1016/j.mechatronics.2011.02.007. URL: http://www.sciencedirect.com/science/article/pii/S0957415811000316.

[162] Matthias Nieuwenhuisen et al. "Collaborative object picking and delivery with a team of micro aerial vehicles at mbzirc". In: *European Conference on Mobile Robots (ECMR)*. IEEE. 2017, pp. 1–6.

[163] A. Nikou et al. "A Nonlinear Model Predictive Control scheme for cooperative manipulation with singularity and collision avoidance". In: *2017 25th Mediterranean Conference on Control and Automation (MED)*. 2017, pp. 707–712.

[164] S Norouzi Ghazbi et al. "QUADROTORS UNMANNED AERIAL VEHICLES: A REVIEW." In: *International Journal on Smart Sensing & Intelligent Systems* 9.1 (2016).

[165] *OGRE - Open Source 3D Graphics Engine*. http://www.ogre3d.org/. 2015.

[166] P. Ogren and N. E. Leonard. "A convergent dynamic window approach to obstacle avoidance". In: *IEEE Transactions on Robotics* 21.2 (2005), pp. 188–195. ISSN: 1552-3098. DOI: 10.1109/TRO.2004.838008.

[167] John-Paul Ore et al. "Autonomous aerial water sampling". In: *Journal of Field Robotics* 32.8 (2015), pp. 1095–1113.

[168] M. Orsag et al. "Stability control in aerial manipulation". In: *American Control Conference (ACC), 2013*. 2013, pp. 5581–5586.

[169] Matko Orsag, Christopher Korpela, and Paul Oh. "Modeling and Control of MM-UAV: Mobile Manipulating Unmanned Aerial Vehicle". English. In: *Journal of Intelligent and Robotic Systems* 69.1-4 (2013), pp. 227–240. ISSN: 0921-0296. DOI: 10.1007/s10846-012-9723-4. URL: http://dx.doi.org/10.1007/s10846-012-9723-4.

[170] Matko Orsag, Christopher Korpela, and Paul Oh. "Modeling and Control of MM-UAV: Mobile Manipulating Unmanned Aerial Vehicle". In: *Journal of Intelligent & Robotic Systems* 69.1 (2013), pp. 227–240. ISSN: 1573-0409. DOI: 10.1007/s10846-012-9723-4. URL: https://doi.org/10.1007/s10846-012-9723-4.

[171] Matko Orsag et al. "Dexterous aerial robotsâĂŤMobile manipulation using unmanned aerial systems". In: *IEEE Transactions on Robotics* 33.6 (2017), pp. 1453–1466.

[172] Razvan Pascanu et al. "How to Construct Deep Recurrent Neural Networks". In: *International Conference on Learning Representations ICLR* (2014). URL: https://arxiv.org/abs/1312.6026.

[173] M. Egmont Petersen, D. de Ridder, and H. Handels. "Image processing with neural networks a review". In: *Pattern Recognition* 35.10 (2002), pp. 2279 –2301. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(01)00178-9. URL: http://www.sciencedirect.com/science/article/pii/S0031320301001789.

[174] Robert Platt et al. "Efficient planning in non-gaussian belief spaces and its application to robot grasping". In: *Robotics Research*. Springer, 2017, pp. 253–269.

[175] M M Polycarpou. "Stable adaptive neural control scheme for nonlinear systems". In: *IEEE Transactions on Automatic Control* 41.3 (1996), pp. 447–451. DOI: 10.1109/9.486648.

[176] Dean A Pomerleau. *Alvinn: An autonomous land vehicle in a neural network*. Tech. rep. DTIC Document, 1989.

[177] Stephen B Pope. "Algorithms for ellipsoids". In: *Cornell University Report No. FDA* (2008), pp. 08–01.

[178] P. E. I. Pounds, D. R. Bersak, and A. M. Dollar. "Stability of small-scale UAV helicopters and quadrotors with added payload mass under PID control". In: *Autonomous Robots* 33.1-2 (2012), pp. 129–142.

[179] Paul EI Pounds, Daniel R Bersak, and Aaron M Dollar. "The yale aerial manipulator: grasping in flight". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 2974–2975.

[180] P.E.I. Pounds, D.R. Bersak, and A.M. Dollar. "The Yale Aerial Manipulator: Grasping in flight". In: *IEEE International Conference on Robotics and Automation (ICRA), 2011*. 2011, pp. 2974 –2975. DOI: 10.1109/ICRA.2011.5980477.

[181] Stephen Prajna, Antonis Papachristodoulou, and Pablo A Parrilo. "Introducing SOSTOOLS: A general purpose sum of squares programming solver". In: *IEEE Conference on Decision and Control (CDC)*. Vol. 1. IEEE. 2002, pp. 741–746.

[182] Samuel Prentice and Nicholas Roy. "The belief roadmap: Efficient planning in belief space by factoring the covariance". In: *The International Journal of Robotics Research* 28.11-12 (2009), pp. 1448–1465.

[183] C. J. Pretorius, M. C. du Plessis, and J. W. Gonsalves. "A comparison of neural networks and physics models as motion simulators for simple robotic evolution". In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. 2014, pp. 2793–2800. DOI: 10.1109/CEC.2014.6900553.

[184] *Prime Air*. http://www.amazon.com/b?node=8037720011.

[185] Demetri Psaltis, Athanasios Sideris, and Alan A Yamamura. "A multilayered neural network controller". In: *IEEE control systems magazine* 8.2 (1988), pp. 17–21.

[186] Dimitris C Psichogios and Lyle H Ungar. "A hybrid neural network-first principles approach to process modeling". In: *American Institute of Chemical Engineers AIChE Journal* 38.10 (1992), pp. 1499–1511. DOI: 10.1002/aic.690381003. URL: http://onlinelibrary.wiley.com/doi/10.1002/aic.690381003/abstract.

[187] G V Puskorius, L A Feldkamp, and L I Davis. “Dynamic neural network methods applied to on-vehicle idle speed control”. In: *Proceedings of the IEEE* 84.10 (1996), pp. 1407–1420. DOI: 10.1109/5.537107.

[188] S Joe Qin and Thomas A Badgwell. “An overview of nonlinear model predictive control applications”. In: *Nonlinear model predictive control*. Springer, 2000, pp. 369–392.

[189] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe. 2009, p. 5.

[190] Guilherme V Raffo, Manuel G Ortega, and Francisco R Rubio. “Backstepping/nonlinear HâĹđ control for path tracking of a quadrotor unmanned aerial vehicle”. In: *American Control Conference, 2008*. IEEE. 2008, pp. 3356–3361.

[191] Sasa V Rakovic et al. “Invariant approximations of the minimal robust positively invariant set”. In: *IEEE Transactions on Automatic Control* 50.3 (2005), pp. 406–410.

[192] SV Rakovic et al. “Computation of invariant sets for piecewise affine discrete time systems subject to bounded disturbances”. In: *IEEE Conference on Decision and Control (CDC)*. Vol. 2. IEEE. 2004, pp. 1418–1423.

[193] SV Rakovic et al. “Simple robust control invariant tubes for some classes of nonlinear discrete time systems”. In: *IEEE Conference on Decision and Control (CDC)*. IEEE. 2006, pp. 6397–6402.

[194] Elon Rimon and Daniel E Koditschek. “Exact robot navigation using artificial potential functions”. In: *Robotics and Automation, IEEE Transactions on* 8.5 (1992), pp. 501–518.

[195] Isabelle Rivals et al. “Real-time control of an autonomous vehicle: a neural network approach to the path following problem”. In: *5th International Conference on Neural Networks and their Applications*. Citeseer, 1993, pp. 219–229.

[196] Douglas G Robertson, Jay H Lee, and James B Rawlings. “A moving horizon-based approach for least-squares estimation”. In: *AIChE Journal* 42.8 (1996), pp. 2209–2224.

[197] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[198] F. Ruggiero et al. "Impedance control of VToL UAVs with a momentum-based external generalized forces estimator". In: *IEEE International Conference on Robotics and Automation (ICRA), 2014*. 2014, pp. 2093–2099. DOI: 10.1109/ICRA.2014.6907146.

[199] T Ryan and H Jin Kim. "LMI-Based Gain Synthesis for Simple Robust Quadrotor Control". In: *IEEE Transactions on Automation Science and Engineering* 10.4 (2013), pp. 1173–1178.

[200] Jihan Ryu. "State and parameter estimation for vehicle dynamics control using GPS". PhD thesis. Stanford University, 2004, p. 125.

[201] Inkyu Sa and Peter Corke. "System identification, estimation and control for a cost effective open-source quadcopter". In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 2202–2209. ISBN: 1050-4729.

[202] I M Salameh, E M Ammar, and T A Tutunji. "Identification of quadcopter hovering using experimental data". In: *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*. IEEE. 2015, pp. 1–6.

[203] Rainer Sandau. "Status and trends of small satellite missions for Earth observation". In: *Acta Astronautica* 66.1 (2010), pp. 1–12.

[204] JÃijrgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85 –117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0893608014002135.

[205] Alexander T Schwarm and Michael Nikolaou. "Chance-constrained model predictive control". In: *AIChE Journal* 45.8 (1999), pp. 1743–1752.

[206] David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[207] Ravid Shwartz-Ziv and Naftali Tishby. "Opening the black box of deep neural networks via information". In: *arXiv preprint arXiv:1703.00810* (2017).

[208] Sebastian Siebert and Jochen Teizer. "Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system". In: *Automation in Construction* 41 (2014), pp. 1–14.

[209] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[210] Sumeet Singh et al. “Robust online motion planning via contraction theory and convex optimization”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2017).

[211] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*. Vol. 199. 1. Prentice hall Englewood Cliffs, NJ, 1991.

[212] *software-link*. `https://github.com/jhu-asco/aerial_autonomy`. 2018.

[213] Matthew Spenko et al. “Hazard avoidance for high-speed mobile robots in rough terrain”. In: *Journal of Field Robotics* 23.5 (2006), pp. 311–331.

[214] Jacob Steinhardt and Russ Tedrake. “Finite-time regional verification of stochastic non-linear systems”. In: *The International Journal of Robotics Research* 31.7 (2012), pp. 901–923.

[215] Stefan Streif et al. “Robust nonlinear model predictive control with constraint satisfaction: A relaxation-based approach”. In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 11073–11079.

[216] A. Suarez, G. Heredia, and A. Ollero. “Lightweight compliant arm with compliant finger for aerial manipulation and inspection”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4449–4454. DOI: `10.1109/IROS.2016.7759655`.

[217] Alessandro Tasora, Dan Negrut, and Mihai Anitescu. “GPU-Based Parallel Computing for the Simulation of Complex Multibody Systems with Unilateral and Bilateral Constraints: An Overview”. English. In: *Multibody Dynamics*. Ed. by Krzysztof Arczewski et al. Vol. 23. Computational Methods in Applied Sciences. Springer Netherlands, 2011, pp. 283–307. ISBN: 978-90-481-9970-9. DOI: `10.1007/978-90-481-9971-6_14`. URL: `http://dx.doi.org/10.1007/978-90-481-9971-6_14`.

[218] *TensorFlow*. `https://www.tensorflow.org/`. 2017.

[219] E. Theodorou, Y. Tassa, and E. Todorov. “Stochastic Differential Dynamic Programming”. In: *Proceedings of the 2010 American Control Conference*. 2010, pp. 1125–1132. DOI: `10.1109/ACC.2010.5530971`.

[220] J. Thomas et al. “Toward image based visual servoing for aerial grasping and perching”. In: *IEEE International Conference on Robotics and Automation (ICRA), 2014*. 2014, pp. 2113–2118. DOI: `10.1109/ICRA.2014.6907149`.

[221] Justin Thomas et al. “Avian-inspired grasping for quadrotor micro UAVs”. In: *American Society of Mechanical Engineers(ASME) 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers. 2013, V06AT07A014–V06AT07A014.

[222] Mark M Tobenkin, Ian R Manchester, and Russ Tedrake. “Invariant funnels around trajectories using sum-of-squares programming”. In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 9218–9223.

[223] A. Torre et al. “A prototype of aerial manipulator”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2012, pp. 2653–2654. DOI: 10.1109/IROS.2012.6386279.

[224] E Velenis and P Tsiotras. “Minimum time vs maximum exit velocity path optimization during cornering”. In: *2005 IEEE international symposium on industrial electronics* (2005), pp. 355–360.

[225] David Vissière, Dong Eui Chang, and Nicolas Petit. “Experiments of trajectory generation and obstacle avoidance for a UGV”. In: *Proc. of the 2007 American Control Conference*. 2007.

[226] Ilolger Voos. “Nonlinear control of a quadrotor micro-UAV using feedback-linearization”. In: *Mechatronics, 2009. ICM 2009. IEEE International Conference on*. IEEE. 2009, pp. 1–6.

[227] Assoc.Prof.Dr.Jompob Waewsak et al. “Estimation of Monthly Mean Daily Global Solar Radiation over Bangkok, Thailand Using Artificial Neural Networks”. In: *Energy Procedia* 57 (2014). DOI: 10.1016/j.egypro.2014.10.103.

[228] Eric A Wan and Rudolph Van Der Merwe. “The unscented Kalman filter for nonlinear estimation”. In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. Ieee. 2000, pp. 153–158.

[229] Youyi Wang, Lihua Xie, and Carlos E de Souza. “Robust control of a class of uncertain nonlinear systems”. In: *Systems & Control Letters* 19.2 (1992), pp. 139–149.

[230] Stephen J Wright. “Efficient Convex Optimization for Linear MPC”. In: *Handbook of Model Predictive Control*. Springer, 2019, pp. 287–303.

[231] Rong Xu and Ümit Özgüner. “Sliding mode control of a quadrotor helicopter”. In: *2006 45th IEEE Conference on Decision and Control*. IEEE. 2006, pp. 4957–4962.

[232] Shuyou Yu et al. “Tube MPC scheme based on robust control invariant set with application to Lipschitz nonlinear systems”. In: *Systems & Control Letters* 62.2 (2013), pp. 194–200.

[233] Wen Yu and Xiaoou Li. “Some new results on system identification with dynamic neural networks”. In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 412–417. DOI: 10.1109/72.914535.

[234] U Yuzgec, Y Becerikli, and M Turker. “Dynamic Neural-Network-Based Model-Predictive Control of an Industrial Baker #x0027;s Yeast Drying Process”. In: *IEEE Transactions on Neural Networks* 19.7 (2008), pp. 1231–1242. DOI: 10.1109/TNN.2008.2000205.

[235] Tianhao Zhang et al. “Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search”. In: *arXiv preprint arXiv:1509.06791* (2015). URL: https://arxiv.org/abs/1509.06791.

[236] Yudong Zhang and Lenan Wu. “Stock market prediction of S&P 500 via combination of improved BCO approach and BP neural network”. In: *Expert Systems with Applications* 36.5 (2009), pp. 8849 –8854. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2008.11.028. URL: http://www.sciencedirect.com/science/article/pii/S095741740800852X.

[237] Kemin Zhou and John Comstock Doyle. *Essentials of robust control*. Vol. 104. Prentice hall Upper Saddle River, NJ, 1998.

Biography

Gowtham Garimella was born in 1991 in Visakhapatnam a city in southern part of India.

Gowtham did his undergraduate in Indian Institute of Technology Bombay (IIT Bombay). He majored in Mechanical Engineering and graduated with honors and a CGPA of 9.63 out of 10 points. After his undergraduation, he joined Johns Hopkins University in 2012 as a Masters student majoring in Mechanical Engineering. He then moved on to pursue a Ph.D. in Mechanical Engineering in 2013 with a special focus on planning and controls for robotic systems.

During his Ph.D., Gowtham worked as a teaching assistant for several robotics courses such as Nonlinear control and planning, Optimal control and Introduction to robotics etc. He received Creel family teaching award for his service as an excellent teaching assistant in 2014 and 2015.

In the later years of Ph.D., Gowtham worked as an intern at a self driving car startup called Zoox. He worked on system identification of the lower-level systems and machine learning applications to controlling the self driving car.

After graduation, Gowtham began working as a full-time employee at Zoox.