Motion Planning for Aerial Manipulation in Constrained Environments

by

Soham Sudhir Patkar

A dissertation submitted to The Johns Hopkins University
in conformity with the requirements for the degree of
Master of Science and Engineering in Robotics

Baltimore, Maryland
May, 2018

Soham S Patkar © 2018
All rights reserved
Abstract

This thesis deals with trajectory planning and control of aerial vehicles equipped with robotic manipulators operating in constrained indoor environments. Aerial manipulators are useful in applications which require manipulation/inspection in regions outside the immediate workspace of the vehicle. These include picking/placing an object from a tall shelf, inspecting a tower or a bridge, and search and rescue activities.

The proposed approach exploits the differentially flat nature of the quadrotor dynamics to solve the planning and trajectory tracking problem. We use trajectory optimization techniques to generate an obstacle-free trajectory and a Model Predictive Control (MPC) formulation to track it in real-time. Lastly, we test the proposed algorithms in a simulation environment and the corresponding results are presented.
Acknowledgments

I would like to thank Marin Kobilarov for his constant guidance and support throughout my time at Johns Hopkins University.

I would also like to thank my lab mates at the Autonomous Systems, Control and Optimization (ASCO) lab, Gowtham Garimella, Matthew Sheckells and Subransu Mishra for their contribution in my work and the constant feedback during my time working in ASCO.

Finally I would like to thank my parents and my sister for their constant support and undying faith in me.
# Table of Contents

Table of Contents iv

List of Figures vi

1 Introduction and Related Work 1

1.1 Related Work .................................................... 2

1.2 Contribution and Organization ................................. 3

2 Overview of Concepts 4

2.1 Trajectory Optimization ......................................... 4

2.2 Model Predictive Control ...................................... 5

2.3 Differentially Flat Systems .................................... 6

3 Trajectory Generation and Control 7

3.1 System Dynamics and Flat Outputs ......................... 7

3.2 Obstacle Formulation and Checking ........................ 9

3.3 Trajectory Generation ......................................... 11
3.3.1 Numerical Optimization ................................. 13
3.4 Trajectory Tracking Controller ............................ 14
  3.4.1 Numerical Optimization ............................... 15

4 Experimental Setup and Validation ..................... 17
  4.1 Simulation Environment ................................. 17
  4.2 Software and Control Architecture .................... 20
  4.3 Experimental Parameters ............................... 21
  4.4 Results .................................................. 22

5 Discussion and Conclusion ............................... 29
# List of Figures

3.1 Collision model of the system ........................................... 10
3.2 Scene Collision Model ...................................................... 10
4.1 Simulation environment in Gazebo ................................. 18
4.2 DJI Matrice quadrotor with manipulator .................. 19
4.3 Custom fabricated 2-dof robotic manipulator .............. 19
4.5 Control Architecture ...................................................... 21
4.6 Side view of planned trajectory for scenario 1 ................. 24
4.7 Top View of the Planned Trajectory for scenario 1 ........ 25
4.8 Planned Trajectory for scenario 2 .................................. 26
4.9 Validation using backstepping controller ..................... 27
4.10 MPC Trajectory Tracking results ................................. 28
Chapter 1

Introduction and Related Work

An aerial manipulation system consists of an aerial vehicle like a quadrotor with a robotic manipulator attached underneath. These systems are capable of interacting with objects that are outside the envelope of the quadrotor. Such systems have peaked interest lately particularly due to their applications in package delivery (Amazon Prime Air). Some additional applications include inspection or servicing in difficult-to-reach environments like high altitude towers, bridges, and rooftops (Kespry). The highly dexterous nature of the system and the dynamics of the interaction between the aerial vehicle and the manipulator make this a challenging planning and control problem. This thesis explores optimal control techniques to solve this problem by exploiting the differentially flat nature of the quadrotor dynamics.
1.1 Related Work

The dynamics of aerial manipulation systems has been studied extensively and numerous feedback control laws have been developed for the same. The ARCAS (Aerial Robotic Co-operative Assembly System) (Kondak et al.) and the Mobile Manipulating Unmanned Aerial Vehicle (MM-UAV) (Orsag, Korpela, and Oh) projects first explored the aerial manipulation problem for unmanned helicopters and small quadrotors respectively. Most of these works use a full multi-body system (MBS) to model the dynamics. Khalifa et al. and Bazylev et al. do trajectory tracking and control using feedback linearization and applying reactive torques to counter the manipulator effects. Kobilarov uses backstepping techniques for doing trajectory control for such systems, while Heredia et al. use a Variable-Parameter Integral Backstepping (VPIB) controller. Bazylev et al., 2014a and Caccavale et al., 2014 attempt to use adaptive control techniques to solve this problem. More recently, there has been work to use Model Predictive Control techniques for planning optimal trajectories for such systems by Garimella and Kobilarov, 2015.

Counter to this, another approach is to decouple the dynamics of the quadrotor and manipulator, treating the interaction forces as disturbances. Ruggiero et al., 2015 is a prior attempt to decouple the dynamics of such a system and to adjust the center of mass to compensate for the disturbances. The MBS model is highly nonlinear in nature which makes the use of optimal control techniques computationally expensive.

There is also some prior work in the field of exploiting the differentially flat
nature of the quadrotor dynamics. Flores and Milam, 2006 discuss the general approach to trajectory generation for differentially flat systems and mention applications to aerial systems. Sheckells, Garimella, and Kobilarov, 2016 use differential flatness to find an optimal trajectory for a visual servoing task. Sreenath and Kumar, 2013 computes flat outputs for a system having an aerial vehicle with a mass suspended from a cable. Yüksel, Buondonno, and Franchi, 2016 describe the control of a quadrotor with multiple parallel manipulators using flat outputs.

Although there is extensive work in the field of control of such systems, the work in the collision-free planning of these systems is sparse. Rossi et al., 2017 is a recent work using quadratic programming for collision-free trajectories. Ragel et al., 2015 compares various sampling-based techniques to perform this task.

1.2 Contribution and Organization

This thesis attempts to solve the planning and trajectory tracking problem by exploiting the differentially flat nature of a simplified version of the system using Trajectory Optimization and Model Predictive Control (MPC) techniques. The simplified system decouples the dynamics of the aerial vehicle and the manipulator and considers only kinematic coupling. The thesis is organized as follows. Chapter 2 gives an brief overview of the concepts that are central to the approach discussed further. Chapter 3 states the formulation of the problem and discusses numerical techniques to solve this problem. Chapter 4 describes the experimental setup and show the results obtained. Chapter 5 discusses the results obtained, conclusions derived from them and avenues for future research.
Chapter 2

Overview of Concepts

2.1 Trajectory Optimization

Trajectory optimization is the process of designing a trajectory for a system with given dynamics that minimizes (or maximizes) some measure of performance while satisfying a set of constraints. (Trajectory Optimization, Wikipedia, The Free Encyclopedia) The cost is expressed in the form of a functional i.e. a function of other functions, where the other functions are the state and control trajectories. It is expressed as:

\[ J(x(\cdot), u(\cdot), t) = \int_{t_0}^{t_f} \mathcal{L}(x(t), u(t), t)dt \quad (2.1) \]

where \( x(\cdot) : \mathbb{R} \rightarrow \mathbb{R}^n \) and \( u(\cdot) : \mathbb{R} \rightarrow \mathbb{R}^m \) are the state and control trajectories and \( \mathcal{L}(x, u, t) \) is the function we want to minimize along the trajectory. In addition, one might want to add terminal cost which penalizes some function of the final state of the trajectory. It can be expressed as \( \phi(x(t_f), t_f) \). The dynamics are expressed as \( \dot{x} = f(x, u, t) \). Let \( \psi(x_f, t_f) = 0 \) be the boundary condition (or terminal constraint) and \( c(x, u, t) \leq 0 \) be the path constraints. In the most general form, the final time \( t_f \)
is also free and included as an optimization variable. The general optimal control problem can be written as:

$$\min_u J(x(\cdot), u(\cdot), t) = \phi(x(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{L}(x(t), u(t), t) dt$$  \hspace{1cm} (2.2)$$

subject to:

$$\dot{x} = f(x(t), u(t), t), \quad x(0) = x_0, \quad t_f \text{ free}$$  \hspace{1cm} (2.3)$$

$$\psi(x(t_f), t_f) = 0$$  \hspace{1cm} (2.4)$$

$$c(x, u, t) \leq 0$$  \hspace{1cm} (2.5)$$

### 2.2 Model Predictive Control

Model Predictive Control (MPC) is an advanced control technique that utilizes a reasonably accurate model and state feedback information to control a system. (Model Predictive Control, Wikipedia, The Free Encyclopedia). At each control interval, an MPC algorithm attempts to optimize future system behavior by computing an optimal trajectory starting from the current system state. The first input in the optimal trajectory is then applied to the system, and the entire calculation is repeated at subsequent control intervals. At each control interval we essentially solve an optimal control problem over a given horizon. For a robotic system this can be either trying to drive the robot to a given goal state or trying to track a reference trajectory.
2.3 Differentially Flat Systems

An important class of systems for which trajectory generation is particularly easy are the so-called differentially flat systems. A system is differentially flat if one can find a set of outputs (equal to the number of inputs) which completely determine the whole state and the inputs without the need to integrate the system. More formally, a system with state $x \in \mathbb{R}^n$ and inputs $u \in \mathbb{R}^m$ is differentially flat if one can find outputs $y \in \mathbb{R}^m$ of the form:

$$y = h(x, u, \dot{u}, \ddot{u}, ..., u(a))$$

such that:

$$x = \varphi(y, \dot{y}, \ddot{y}, ..., y(b))$$

$$u = \alpha(y, \dot{y}, \ddot{y}, ..., y(c))$$

where $h$, $\varphi$ and $\alpha$ are bijective functions. If such outputs exist, one can solve the trajectory generation problem in the flat output space. The advantage of flat output space is that the differential constraints of the system are now linear which make the optimization less computationally expensive. The resultant optimal trajectory can be recovered from $\varphi$ and $\alpha$. 
Chapter 3

Trajectory Generation and Control

3.1 System Dynamics and Flat Outputs

The system is a quadrotor model with a $n$-DOF manipulator attached to it. The state of the aerial manipulation system is given by $(p, R, v, \omega, q_1, ..., q_n)$ where $p \in \mathbb{R}^3$ is the center of mass (COM) position, $R \in SO(3)$ is the base orientation, $v \in \mathbb{R}^3$ is the COM velocity and $\omega \in \mathbb{R}^3$ is the angular velocity of the quadrotor with respect to an inertial frame, and $q_1, ..., q_n$ are the $n$ joint states of the manipulator. The control inputs to this system are $\tau = (\tau_x, \tau_y, \tau_z)$ the 3 body torques about the body $x, y, z$ axes of the quadrotor, the vertical thrust along the body $z$-axis $T$, and the $n$ joint velocities of the manipulator $\dot{q}_1, ..., \dot{q}_n$. We assume that the joint torques applied by the lower level controller of the manipulator are small enough that they have negligible effect on the dynamics of the quadrotor. In short, we assume that the coupling between the quadrotor and the manipulator is purely kinematic. That
being said, the dynamics of the system can be written as:

\[
\dot{p} = v \quad (3.1)
\]

\[
\dot{v} = \frac{1}{m} Re_3 T + g \quad (3.2)
\]

\[
\dot{R} = R\dot{\omega} \quad (3.3)
\]

\[
\dot{\omega} = J^{-1}[J\omega \times \omega + \tau] \quad (3.4)
\]

\[
\dot{q}_i = \dot{q}_i \quad (3.5)
\]

For the quadrotor, the flat outputs are given by \([p, \gamma]\) the position and yaw of the quadrotor. The position and velocity can be recovered directly from \(p\). We can recover the remaining states and controls of the quadrotor using the following
relations:

\[ T = ||m(\ddot{p} - g)|| \]  \hspace{1cm} (3.6)

\[ R_z = m(\ddot{p} - g)/T \]  \hspace{1cm} (3.7)

\[ R_y = R_z \times \begin{pmatrix} \cos(\gamma) \\ \sin(\gamma) \\ 0 \end{pmatrix} / ||R_z \times \begin{pmatrix} \cos(\gamma) \\ \sin(\gamma) \\ 0 \end{pmatrix} || \]  \hspace{1cm} (3.8)

\[ R_x = R_y \times R_z \]  \hspace{1cm} (3.9)

\[ \omega_x = -R_y \cdot p^{(3)}/T \]  \hspace{1cm} (3.10)

\[ \omega_y = R_x \cdot p^{(3)}/T \]  \hspace{1cm} (3.11)

\[ \omega_z = \ddot{\gamma}(e_3 \cdot R_z) \]  \hspace{1cm} (3.12)

\[ \dot{\omega}_x = \left( -mR_y \cdot p^{(4)} - \omega_y \omega_z T + 2\omega_x \dot{T} \right) / T \]  \hspace{1cm} (3.13)

\[ \dot{\omega}_x = \left( mR_y \cdot p^{(4)} - \omega_x \omega_z T - 2\omega_y \dot{T} \right) / T \]  \hspace{1cm} (3.14)

\[ \dot{\omega}_z = \dot{\gamma}e_3^T R\dot{\omega}_3 \]  \hspace{1cm} (3.15)

\[ \tau = J\dot{\omega} - J\omega \times \omega \]  \hspace{1cm} (3.16)

### 3.2 Obstacle Formulation and Checking

For obstacle avoidance, we use a simple model where the collision envelope of any body is expressed in the form of a number of overlapping spheres arranged in
specific manner so as to cover the given body. The number and size of these spheres are chosen to balance the computational complexity and conservativeness of the collision model. For our system, we use 4 spheres for the quadrotor, 4 spheres for each arm link, and one sphere for the end effector (Figure 3.1). The positions of the centers of these spheres are propagated using the quadrotor’s position and forward kinematics of the arm. Obstacles like tables and small objects are modeled as an appropriate number overlapping spheres, while taller obstacles like poles can be modeled as axis-aligned cylinders (Figure 3.2).

For collision checking we just make sure that none of the spheres on the quadrotor body or the manipulator intersect with any of the spheres or cylinders of the obstacles. This can be written as:

$$||p^b_i - p^o_j||^2 > (r^b_i - r^o_j)^2 \quad \text{for} \ i = 1, ..., N \ \text{and} \ j = 1, ..., M$$
where \( p_i^b \) is the center and \( r_i^b \) is the radius of the \( i^{th} \) body sphere and \( p_i^o \) is the center and \( r_i^o \) is the radius of the \( j^{th} \) obstacle sphere. \( N \) is the number of spheres on the body and \( M \) is the number of obstacle spheres. For our setup, we have a 2-dof arm so the self-collision checking is done implicitly by joint angle limits. In general, for self-collision one can impose additional body-to-body collision checking.

### 3.3 Trajectory Generation

We now have the resulting system where the state \( x \in \mathbb{R}^{(6+n)} \) is

\[
x = \begin{bmatrix}
p \\
p' \\
p'' \\
p^{(3)} \\
\gamma \\
\dot{\gamma} \\
q_1 \\
\vdots \\
q_n
\end{bmatrix}
\]

and the control inputs \( u \in \mathbb{R}^{(2+n)} \) are

\[
u = \begin{bmatrix}
p^{(4)} \\
\ddot{\gamma} \\
\dot{q}_1 \\
\vdots \\
\dot{q}_n
\end{bmatrix}
\]

The dynamics of the system can be written as:

\[
\dot{x} = Ax + Bu \tag{3.17}
\]
where \( A \in \mathbb{R}^{(6+n) \times (6+n)} \) and \( B \in \mathbb{R}^{(2+n) \times (2+n)} \) are constant sparse matrices that can be derived easily from the dynamics.

Given this new system, we can generate a trajectory that takes the end-effector to a desired pose in the inertial frame, with obstacle avoidance being imposed as path constraints. Let \( g(x(t)) : \mathbb{R}^{(6+n)} \to SE(3) \) be the end-effector pose in the inertial frame. Let \( g_f \in SE(3) \) be the goal end-effector pose in the inertial frame. Let \( c(x, u, t) \leq 0 \) denote the set of constraints on the states and controls; like velocity, torque, thrust limits. We want our system to behave such that it shows little effect of manipulator torques on the quadrotor dynamics. To ensure this we add a quadratic cost on the total control effort over the trajectory. So the optimal control problem can be written as:

\[
\min_u J = \int_{t_0}^{t_f} u(t)^T Q(t) u(t) dt \quad (3.18)
\]

subject to:

\[
\dot{x} = f(x, u, t), \quad x(0) = x_0 \quad (3.20)
\]
\[
g(x(t_f)) = g_f \quad (3.21)
\]
\[
||p_i^b - p_j^o||^2 > (r_i^b - r_j^o)^2 \quad (3.22)
\]
\[
c(x, u, t) \leq 0 \quad (3.23)
\]
3.3.1 Numerical Optimization

To solve the higher level optimization problem we use a well-established numerical technique called Direct Collocation. Collocation has the advantage that the path constraints can be handled very easily as opposed to shooting methods. In collocation, the time horizon is divided into $N$ segments. $x_{0:N} = \{x_0, x_1, ..., x_N\}$ are the states and $u_{0:N} = \{u_0, u_1, ..., u_{N-1}\}$ are the controls corresponding to the time steps. We replace the non-linear dynamics with a implicit finite difference approximation like Runge-Kutta $4^{th}$ order integrator and impose it as a constraint in the optimization.

$$S(x_k, x_{k+1}, u_k, u_{k+1}, t_k) = x_{k+1} - x_k - h_k \cdot rk_4(x_k, x_{k+1}, u_k, u_{k+1}, t_k) = 0 \quad (3.24)$$

where $rk_4$ is the Runge-Kutta integrator and $h_k$ is the time step. The cost itself can be modeled as:

$$J = \sum_{k=0}^{N-1} u_k^T Q_k u_k \quad (3.25)$$
So the optimal control problem can be formulated as a inequality-constrained non-linear optimization problem:

$$\min_{\mathbf{u}} J = \sum_{k=0}^{N-1} u_k^T Q_k u_k$$

subject to:

$$S(x_k, x_{k+1}, u_k, u_{k+1}, t_k) = 0$$

$$||p_i^b - p_j^o||^2 > (r_i^b - r_j^o)^2$$

$$c(x_k, u_k, t_k) \leq 0$$

The optimization can be solved as a Non-Linear Program (NLP), with the NLP variables being $[x_0:N, u_0:N]$, using Sequential Quadratic Programming (SQP) methods with active set techniques for handling path constraints or Interior Point (IP) methods.

### 3.4 Trajectory Tracking Controller

Once the trajectory is generated, we have a lower level MPC formulation to track this trajectory in real time. We choose a horizon $t_h$, and a trajectory of time $t_h$ seconds is generated from the current state that tracks the reference trajectory as closely as possible. This is ensured by having a quadratic cost on the deviation from the reference trajectory. We run the system with the controls from this trajectory for
some time, and then regenerate a $t_h$ second trajectory from the resulting state. This is repeated until convergence. Since the obstacles are stationary, we do not impose obstacle constraints on the lower level controller. The optimal control formulation can be written as:

$$\min_u J = \int_{t}^{t+h} (x - x^{ref})^T W (x - x^{ref}) + (u - u^{ref})^T Q (u - u^{ref}) d\tau$$  \hspace{1cm} (3.32)

subject to:

$$\dot{x} = f(x, u, t),$$ \hspace{1cm} (3.34)

$$c(x, u, t) \leq 0$$ \hspace{1cm} (3.35)

Given the output trajectory in flat output space, the controls are recovered using equations (3.6) - (3.16).

### 3.4.1 Numerical Optimization

Since the lower level controller has no path constraints, we can model the optimization using a numerical technique called Direct Multiple Shooting. Similar to earlier formulation we divide the trajectory into $N$ time steps. In multiple shooting, the control trajectory between two time steps is parametrized by a using e.g. some well-known curve like B-spline or Bezier curve. However, in our case we choose a simple piecewise constant control parametrization since we are directly controlling the snap of the quadrotor position, and a constant snap will give a piecewise cubic velocity profile. Let $\bar{x}_0:N-1$ be the states obtained by integrating the controls. Then
we can impose constraints on the optimization that \( \bar{x}_k - x_{k-1} = 0 \). The cost can be written as \( J = \frac{1}{2} h(u)^T h(u) \) where \( h \) is given by:

\[
h(u) = \begin{bmatrix}
\sqrt{W_0} (x_0 - x_0^{ref}) \\
\sqrt{Q_0} (u_0 - u_0^{ref}) \\
\vdots \\
\sqrt{W_k} (x_k - x_k^{ref}) \\
\sqrt{Q_k} (u_k - u_k^{ref}) \\
\vdots \\
\sqrt{W_{N-1}} (x_{N-1} - x_{N-1}^{ref}) \\
\sqrt{Q_{N-1}} (u_{N-1} - u_{N-1}^{ref})
\end{bmatrix}
\]  

(3.36)

So the optimization can be written as:

\[
\min_{u} J = \frac{1}{2} h(u)^T h(u)
\]  

(3.37)

subject to:

\[
\begin{bmatrix}
\bar{x}_0 - x_1 \\
\bar{x}_1 - x_2 \\
\vdots \\
\bar{x}_{N-2} - x_{N-1}
\end{bmatrix} = 0
\]  

(3.39)
Chapter 4

Experimental Setup and Validation

4.1 Simulation Environment

The dynamics is simulated in the Gazebo environment with a quadrotor-manipulator system created as a plugin to emulate the inertial and geometric properties of the DJI Matrice 100 quadrotor (Figure 4.2) and a custom fabricated 2-DOF using 2 Dynamixel MX-28 smart servo motors (Figure 4.3). The quadrotor has an approximate diameter of 1.1 meters and is around 30 cm high. The arm links are 25 cm and 30 cm each. The collision model is formulated according to this geometry. The plugin also emulates a controller provided by the DJI Software Development Kit that can provide roll-pitch-yaw rate-thrust commands to the quadrotor and a cascaded PID controller for the manipulator to achieve desired joint angles.
Figure 4.1: Simulation environment in Gazebo: The green cylinder is the object to be picked up kept on the blue table. The red poles are obstacles the quadrotor needs to avoid.
Figure 4.2: DJI Matrice quadrotor with manipulator

Figure 4.3: Custom fabricated 2-dof manipulator

Figure 4.4: Simulated system in Gazebo environment
4.2 Software and Control Architecture

The numerical optimization is setup in an ACADO Environment in C++. ACADO Toolkit is a software environment and algorithm collection for automatic control and dynamic optimization (Houska, Ferreau, and Diehl, 2011). It provides a general framework for using a great variety of algorithms for direct optimal control, including model predictive control, state and parameter estimation and robust optimization.

The trajectory generator assumes a knowledge of a map of the environment and obstacle locations. Given the map, we represent the obstacles as spheres and cylinders as mentioned in section 3.2. The optimization in section 3.3 is formulated for ACADO in C++ by passing these values and velocity, torque, and joint limits. The output of the optimization is an collision-free trajectory that is passed as reference to the trajectory tracking controller.

The trajectory tracking controller has a similar setup using ACADO in C++. The controller communicates with Gazebo using Robot Operating System (ROS) middleware. The controller receives pose sensor data from the Gazebo plugin which the optimization uses as an initialization for the next optimization step. Since no information is available about the other states, we make a naive assumption that the output from the previous optimization has been achieved, and that is used to initialize the next step of the optimization. The tracking controller is run at 10 Hz. Though the optimization can run much faster, the output was found to be unstable at high frequencies. To make up for this, instead of passing roll-pitch-yaw-thrust
commands directly to the lower level controller, we add another control layer that tries to achieve a goal velocity and yaw by commanding roll-pitch-yaw-thrust. This is a PI controller that runs at 50 Hz to which the goal velocity and yaw is provided from the resulting state of the optimization. The controller recovers the roll-pitch-yaw-thrust command using equations (3.6)-(3.9), and passes the command to the lower level controller in Gazebo. Figure 4.5 shows the control architecture and the data flow among the components.

4.3 Experimental Parameters

Though it is possible for the trajectory generation to optimize over the final time, we choose it manually to make the formulation simpler for the collocation problem. We consider 2 scenarios for the planning problem: one in which the quadrotor has to navigate through a narrow gap and another in which the arm has to navigate around an obstacle. We choose $t_f = 10$ and for the trajectory generation. We constrain the speed to be less than 1 m/s in each direction and yaw rate to be less than 1 rad/s. There is a constraint on the thrust to be less than 20 N and roll and
pitch rates are constrained to be less than \( \pi \text{ rad/s} \). We divide the trajectory into 40 steps for the collocation problem. The reference trajectory is recovered as a set of 40 states and controls with a time-step of 0.4 seconds. For the trajectory tracking controller, we roll out a 1 second trajectory, divided into 10 multiple-shooting steps. The controller keeps track of time passed since the controller started, and considers a 1 second reference trajectory accordingly.

4.4 Results

The goal of trajectory generation was to plan an obstacle free trajectory for the environment shown in Figure 4.1. The environment has 2 poles separated by 1.5 meters, approximately mid-way between the initial and final position. The quadrotor starts in front of one of the 2 poles at a takeoff height of 1.5 meters. The goal is to reach a 10 cm high object placed on a table about 0.5 meters high, which is about 3 meters away from the initial quadrotor position. We also assume the scenario takes place in small room which is 3 meters wide and about 5 meters long. The desired outcome is for the quadrotor to navigate between the 2 poles and reach the goal end-effector position without the end-effector colliding with the table.

The trajectory generation was also tested in a scenario where the system starts with the end-effector under an obstacle and the goal end-effector above it. Figure 4.8 shows that the resulting trajectory guides the arm around the obstacle and reaches the goal.

The resulting trajectories are shown in Figures 4.7 and 4.6. Figure 4.7 shows that
the quadrotor successfully plans a trajectory that will guide it through the narrow gap between the poles. Figure 4.6 shows that the resultant trajectory also makes sure that the end-effector of the manipulator does not collide with the table. To validate the fact that the trajectory is feasible and can be tracked by the system, we also try to track this trajectory using a backstepping controller. The results in 4.9 show that the generated trajectory is feasible and can be achieved by the system.

The reference from the first scenario is used for the trajectory tracking simulation since there is significant motion of both the quadrotor and the arm in this scenario. The trajectory generated is passed as a reference trajectory to the trajectory tracking controller. The results from the simulation are summarised in Figure 4.10. The plots show the reference trajectory generated, the outputs of the MPC controller and the actual states achieved by the system. The plots show that certain state oscillate around the reference trajectory but eventually converge near the goal state.
Figure 4.6: Side View: The quadrotor trajectory is shown in red while the end-effector trajectory is shown in green.
Figure 4.7: Top View: The quadrotor trajectory is shown in red while the end-effector trajectory is shown in green.
Figure 4.8: Second Scenario: The quadrotor trajectory is shown in red while the end-effector trajectory is shown in green
Figure 4.9: Validation of trajectory: The trajectory tracked using the backstepping controller. The dashed red line shows the reference and the blue line shows the actual trajectory achieved by the system.
Figure 4.10: MPC Trajectory Tracking: The output trajectory achieved by the system. The red dashed line represents the reference trajectory, green line represents the output from the MPC controller while the blue line is the actual trajectory of the system.
Chapter 5

Discussion and Conclusion

This thesis explores the possibilities of using the differentially flat nature of the simplified model to solve the trajectory planning and control problem for an aerial manipulation system. The results obtained for the trajectory generation show that we can successfully obtain obstacle free trajectories for the system by planning on flat output space. This was demonstrated for two scenarios: one in which the quadrotor has to pass through a narrow gap and another in which the system needs to navigate the manipulator around an obstacle. The generated trajectory was validated using a backstepping controller and the system was shown to be capable of tracking it. Using the trajectory tracking controller, however, led the system to oscillate around the reference trajectory. More specifically the controller was found to be unstable at high frequency, and sensitive to sensor noise and small deviation of initial state from the reference. Possible reasons for this can be that the orientation and angular velocities are obtained implicitly in flat output space using the acceleration and jerk. The mapping to acceleration and angular velocity is possibly sensitive to small variations in acceleration and jerk which cause the
system to be unstable.

Even though the trajectory generation does a decent job, it is still an offline method as it runs at around 1 Hz. One avenue for future research might be to find techniques that can implement real-time trajectory planning in an MPC fashion. Another direction is to try to use the flat outputs of the full MBS model of this system for trajectory generation. Though the applications that motivated this research do not require particularly aggressive trajectories, it might be interesting to generate more aggressive trajectories and find out when the interacting forces start to make significant impact. For the MPC trajectory tracking, it can be seen from the results that modeling of the disturbances caused by the interactions is crucial to a more robust implementation.
References


Quigley, Morgan, Josh Faust, Tully Foote, and Jeremy Leibs. “ROS: an open-source Robot Operating System”. In:


Sreenath, Koushil and Vijay Kumar (2013). “Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots”. In: nn 1.r2, r3.


SOHAM SUDHIR PATKAR
+1-(443)-370-6958   spatkar1@jhu.edu   LinkedIn: https://www.linkedin.com/in/sohampatkar

Robotics masters student experienced in developing C++ based software for localization, motion planning & control of real-time robotic systems

EDUCATION

MSE Robotics
Johns Hopkins University
Relevant Coursework: Machine Learning, Computer Vision, Robot Systems Programming, Sensor-Based Robotics

B.E(Hons) Mechanical Engineering
BITS Pilani, KK Birla Goa Campus

TECHNICAL SKILLS

Programming  C++ & Python in Linux Environment
Platforms  Robot Operating System(ROS), Linux, OpenCV, PyTorch
Other Tools  Git, CMake, gdb, valgrind

RESEARCH EXPERIENCE

Graduate Research Assistant, Autonomous Systems Control & Optimization Lab
January '17 - Present
Research Focus: Motion Planning of a UAV-Manipulator System for aerial manipulation with obstacle avoidance

- Model Predictive Control: Worked on formulation of an MPC planner to generate obstacle-free trajectories and a Receding Horizon Controller to track the trajectory for a UAV-Manipulator system.
- Software framework: Worked on development of a C++ based framework for system integration of UAV hardware with sensors, controllers etc
- Sensor Fusion: Developed C++ based package for localization of a UAV by sensor fusion of GPS, IMU and an optical flow sensor using an EKF.
- Motion Planning: Developed a ROS-based C++ package to plan obstacle free trajectories for a 7-dof manipulator using Sampling-Based Motion Planning (RRT).

CURRICULAR PROJECTS

Visual Odometry using Deep Learning
December '17
- Designed a convolutional neural networks to estimate the movement of a vehicle between 2 consecutive frames.
- Used above network to estimate odometry information of the vehicle.

Deep Learning for Image Classification
November '17 - December '17
- Starting from a linear model, built up a deep neural network for image classification.
- Added normalization and regularization techniques like batch norm, dropout and data augmentation and compared the performance of the network.

3D SLAM using UAV and AR markers
April '17 - May '17
- Developed a ROS package for crude 3D SLAM by relative pose estimation of AR Markers from a drone camera.

Vision-based Pick and Place using a 6-DOF Manipulator
November '16 - December '16
- Developed a controller for pick-and-place operation for a 6-DOF robotic arm using end-effector pose information from a camera.

WORK EXPERIENCE

Teaching Assistant
August '17 - Present
- Robot Systems Programming: Introduction to the ROS Environment and using it for integration of various robotic systems.
- Applied Optimal Control: Formulating models for optimal control of dynamical systems subject to constraints and uncertainty and use numerical techniques like iLQR, DDP, direct shooting etc.