Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System

Chapter *in* Studies in Computational Intelligence · May 2017

DOI: 10.1007/978-3-319-54927-9_1

CITATIONS
99

READS
4,844

4 authors:

Mina Samir Kamel
ETH Zurich
37 PUBLICATIONS **890** CITATIONS
SEE PROFILE

Thomas Stastny
ETH Zurich
36 PUBLICATIONS **389** CITATIONS
SEE PROFILE

Kostas Alexis
ETH Zurich
119 PUBLICATIONS **2,610** CITATIONS
SEE PROFILE

Roland Siegwart
ETH Zurich
797 PUBLICATIONS **35,136** CITATIONS
SEE PROFILE

Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System

Mina Kamel, Thomas Stastny, Kostas Alexis and Roland Siegwart

Abstract In this chapter, strategies for Model Predictive Control (MPC) design and implementation for Unmanned Aerial Vehicles (UAVs) are discussed. This chapter is divided into two main sections. In the first section, modelling, controller design and implementation of MPC for multi-rotor systems is presented. In the second section, we show modelling and controller design techniques for fixed-wing UAVs. System identification techniques are used to derive an estimate of the system model, while state of the art solvers are employed to solve online the optimization problem. By the end of this chapter, the reader should be able to implement an MPC controller to achieve trajectory tracking for both multi-rotor systems and fixed-wing UAVs.

1 Introduction

Aerial robots are gaining great attention recently as they have many advantages over ground robots to execute inspection, search and rescue, surveillance and goods delivery tasks. Depending on the task required to be executed, a multi-rotor system or fixed-wing aircraft might be a more suitable choice. For instance, a fixed-wing aircraft is more suitable for surveillance and large-scale mapping tasks thanks to their long endurance capability and higher speed compared to a multi-rotor system, while for an inspection task that requires flying close to structures to obtain detailed footage a multi-rotor UAV is more appropriate.

Precise trajectory tracking is a demanding feature for aerial robots in general in order to successfully perform required tasks, especially when operating in realistic environment where external disturbances may heavily affect the flight performance

Mina Kamel, Thomas Stastny and Roland Siegwart
Autonomous System Lab, ETH Zurich,
email {fmina, tstastny, rsiegwart}@ethz.ch.

Kostas Alexis
University of Nevada, Reno, email kalexis@unr.edu

and when flying in the vicinity of structure. In this chapter, several model predictive control strategies for trajectory tracking are presented for multi-rotor systems as well as for fixed-wing aircraft. The general control structure followed by this chapter is a cascade control approach, where a reliable and system-specific low-level controller is present as inner loop, and a model-based trajectory tracking controller is running as an outer loop. This approach is motivated by the fact that many critical flight software is running on a separate navigation hardware which is typically based on micro-controllers, such as Pixhawk PX4 and Navio [1, 2] while high level tasks are running on more powerful -but less reliable- on-board computers. This introduces a separation layer to keep critical tasks running despite any failure in the more complex high-level computer.

By the end of this chapter, the reader should be able to implement and test various Model predictive Controller (MPC) strategies for aerial robots trajectory tracking, and integrate these controllers into the Robot Operating System (ROS) [3]. Various implementation hints and practical suggestions are provided in this chapter and we show several experimental results to evaluate the proposed control algorithms on real systems.

In Section 2 the general theory behind MPC is presented, with focus on linear MPC and robust linear MPC. In Section 3 we present the multi-rotor system model and give hints on model identification approaches. Moreover, linear and robust MPC are presented and we show how to integrate these controller into ROS and present experimental validation results. In Section 4 we present a Nonlinear MPC approach for lateral-directional position control of fixed-wing aircraft with implementation hints and validation experiments.

2 Background

2.1 Concepts of Receding Horizon Control

Receding Horizon Control (RHC) corresponds to the historic evolution in control theory that aimed to attack the known challenges of fixed horizon control. Fixed horizon optimization computes a sequence of control actions $\{u_0, u_1, ..., u_{N-1}\}$ over a horizon N and is characterized by two main drawbacks, namely: a) when an unexpected (unknown during the control design phase) disturbance takes place or when the model employed for control synthesis behaves different than the actual system, then the controller has no way to account for that over the computed control sequence, and b) as one approaches the final control steps (over the computer fixed horizon) the control law "gives up trying" since there is too little time left in the fixed horizon to go to achieve a significant objective function reduction. To address these limitations, RHC proposed the alternative strategy of computing the full control sequence, applying only the first step of it and then repeating the whole process

iteratively (receding horizon fashion). RHC strategies are in general applicable to nonlinear dynamics of the form (considering that state update is available):

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{1}$$

where the vector field $f : \mathbb{R}^n \times \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^{n\times 1}$ represents the state vector, and $\mathbf{u} \in \mathbb{R}^{m\times 1}$ the input vector. The general state feedback–based RHC optimization problem takes the following form:

$$\begin{aligned} \min_{z} \quad & F(\mathbf{x}_{t+N}) + \sum_{k=0}^{N-1} ||\mathbf{x}_{t+k} - \mathbf{r}_t||_\ell + ||\mathbf{u}_{t+k}||_\ell \\ s.t. \quad & \mathbf{x}_{t+k+1} = f(\mathbf{x}_{t+k}, \mathbf{u}_{t+k}) \\ & \mathbf{u}_{t+k} \in \mathscr{U}_C \\ & \mathbf{x}_{t+k} \in \mathscr{X}_C \\ & \mathbf{x}_t = \mathbf{x}(t),\ k = 0, ..., 1 \end{aligned} \tag{2}$$

where ℓ denotes some (penalized) metric used for per–stage weighting, $F(\mathbf{x}_{t+N})$ represents the terminal state weighting, $\mathbf{r}_t$ is the reference signal, $\square_{t+k}$ is used to denote the sample (using a fixed sampling time T_s) of a signal at k steps ahead of the current time t, $\square_{t+k+1}$ the next evolution of that, $\mathscr{U}_C$ represents the set of input constraints, $\mathscr{X}_C$ the state constraints and $\mathbf{x}(t)$ is the value of the state vector at the beginning of the current RHC iteration. The solution of this optimization problem leads again to an optimal control sequence $\{\mathbf{u}_t^\star, \mathbf{u}_{t+1}^\star, \mathbf{u}_{t+N-1}^\star\}$ but only the first step of that $\mathbf{u}_t^\star$ is applied while the whole process is then repeated iteratively.

Within this formulation, the term $F(\mathbf{x}_{t+N})$ has a critical role for the closed–loop stability. In particular, it forces the system state to take values within a particular set at the end of the prediction horizon. It is relatively easy to prove stability per local iteration using Lyapunov analysis. In its simplest case, this essentially means that considering the regulation problem ($\mathbf{r}_t = 0$), and a "decrescent" metric ℓ, then the solution of the above optimization problem makes the system stable at $\mathbf{x}_t = \mathbf{0}$, $\mathbf{u}_t = \mathbf{0}$ – that is that a terminal constraint $\mathbf{x}_{t+k} = 0$ is introduced (a simplified illustration is provided in Figure 1). However, the question of global stability is in general not guaranteed. For that one has to consider the problem of introducing both a terminal cost and a terminal constraint for the states [4]. However, general constrained optimization problems can be extremely difficult to solve, and simply adding terminal constraints may not be feasible. Note that in many practical cases, the terminal constraint is not enforced during the control design procedure, but rather verified a posteriori (by increasing the prediction horizon if not satisfied).

Furthermore, one of the most challenging properties of RHC is that of recursive feasibility. Unfortunately, although absolutely recommended from a theoretical standpoint, it is not always possible to construct a RHC that has a–priori guarantee of recursive feasibility, either due to theoretical or practical implications. In general, a RHC strategy lacks recursive feasibility –and is therefore invalidated– even when it is possible to find a state which is feasible, but where the optimal control action

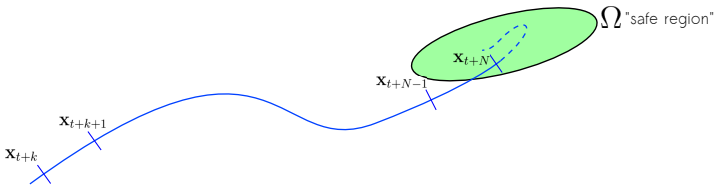


Fig. 1 Illustration of the terminal constraint set (Ω).

moves the state vector to a point where the RHC optimization problem is infeasible. Although a general feasibility analysis methodology is very challenging, for specific cases powerful tools exist. In particular, for the case of linear systems then the Farkas' Lemma [5] in combination with bilevel programming can be used to search for problematic initial states which lack recursive feasibility – thus invalidating an RHC strategy.

2.2 Linear Model Predictive Control

In this subsection we briefly present the theory behind Model Predictive Control for linear systems. We formulate the optimal control problem for linear systems with linear constraints in the input and state variables. Moreover, we discuss the control input properties, stability and feasibility in the case of linear and quadratic cost function. To achieve offset free tracking under model mismatch, we adopt the approach described in [6] where the system model is augmented with additional disturbances state $d(t)$ to capture the model mismatch. An observer is employed to estimate disturbances in steady state. The observer design and the disturbance model will be briefly discussed in this subsection.

$$\begin{aligned} \min_{U} \quad & J_0\left(x_0, U, X_{ref}, U_{ref}\right) \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k + B_d d_k; \\ & d_{k+1} = d_k, \quad k = 0, \ldots, N-1 \\ & x_k \in \mathbb{X}, \quad u_k \in \mathbb{U} \\ & x_N \in \mathbb{X}_N \\ & x_0 = x(t_0), \quad d_0 = d(t_0). \end{aligned} \tag{3}$$

The optimal control problem to achieve offset-free state tracking under linear state and and input constraints is shown in (3), where J_0 is the cost function,$X_{ref}^T = \left[x_{ref,0}^T, \ldots, x_{ref,N}^T\right]^T$ is the reference state sequence, $U = \left[u_0^T, \ldots, u_{N-1}^T\right]^T$ and $U_{ref} = \left[u_{ref,0}^T, \ldots, u_{ref,N-1}^T\right]^T$ are respectively the control input sequence and the steady state input sequence, B_d is the disturbance model and d_k is the external disturbances, $\mathbb{X}$, $\mathbb{U}$ and $\mathbb{X}_N$ are polyhedra. The choice of the disturbance model is not a trivial

task, and depends on the system under consideration and the type of disturbances expected. The optimization problem is defined as

$$\begin{aligned} J_0\left(x_0, U, X_{ref}, U_{ref}\right) = \sum_{k=0}^{N-1} & \left(x_k - x_{ref,k}\right)^T Q_x \left(x_k - x_{ref,k}\right) + \\ & \left(u_k - u_{ref,k}\right)^T R_u \left(u_k - u_{ref,k}\right) + \\ & \left(u_k - u_{k-1}\right)^T R_\Delta \left(u_k - u_{k-1}\right) + \\ & \left(x_N - x_{ref,N}\right)^T P \left(x_N - x_{ref,N}\right), \end{aligned} \tag{4}$$

where $Q_x \succeq 0$ is the penalty on the state error, $R_u \succ 0$ is the penalty on control input error, $R_\Delta \succeq 0$ is a penalty on the control change rate and P is the terminal state error penalty.

In general, stability and feasibility of receding horizon problems are not ensured except for particular cases such as infinite horizon control problems (LQR). When the prediction horizon is limited to N, the stability and feasibility guarantees are disputable. In principle, longer prediction horizon tends to improve stability and feasibility properties of the controller, but the computation effort will increase, and for aerial robot application, fast control action needs to be computed on limited computation power platforms. However, the terminal cost P and terminal constraint $\mathbb{X}_N$ can be chosen such that closed-loop stability and feasibility are ensured [6]. In this chapter we focus more on the choice of terminal weight P as it is easy to compute, while the terminal constraint is generally more difficult and practically stability is achieved with long enough prediction horizon.

Note that in our cost function, we penalize the control input rate Δu_k. This ensures smooth control input and avoids undesired oscillations. In the cost function 4, u_{-1} is the actual control input applied on the system in the previous time step.

As previously mentioned, offset-free reference tracking can be achieved by augmenting the system model with disturbances d_k to capture the modeling error. Assuming that we want to track the system output $y_k = Cx_k$ and achieve steady state offset free tracking $y_\infty = r_\infty$. A simple observer that can estimate such disturbance can be achieved as follows

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \left(C\hat{x}_k - y_{m,k}\right) \tag{5}$$

where $\hat{x}$ and $\hat{d}$ are the estimated state and external disturbances, L_x and L_d are the observer gains and $y_{m,k}$ is the measured output at time k.

Under the assumption of stable observer, it is possible to compute the MPC state at steady state x_{ref} and control input at steady state u_{ref} by solving the following system of linear equations:

$$\begin{bmatrix} A - I & B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_{ref,k} \\ u_{ref,k} \end{bmatrix} = \begin{bmatrix} -B_d \hat{d}_k \\ r_k \end{bmatrix} \tag{6}$$

2.3 Nonlinear Model Predictive Control

Aerial vehicles behavior is better described by a set of nonlinear differential equations. Therefore in this subsection we present the theory behind Nonlinear MPC that exploits the full system dynamics, and generally achieve better performance when it comes to aggressive trajectory tracking. The optimization problem for nonlinear MPC is formulated in Equation (7).

$$\begin{aligned} \min_{U,X} \int_{t=0}^{T} \left\| h(x(t),u(t)) - y_{ref} \right\|_Q^2 dt + \left\| h(x(T)) - y_{N,ref} \right\|_{Q_N}^2 \\ \text{subject to} \quad \dot{x} = f(x(t),u(t)); \\ u(t) \in \mathbb{U} \\ x(t) \in \mathbb{X} \\ x(0) = x(t_0). \end{aligned} \tag{7}$$

A direct multiple shooting technique is used to solve the optimal control problem (7). In this approach the system dynamics are discretized over a time grid $t_0, \ldots, t_N$ within the time intervals $[t_k, t_{k+1}]$. The inequality constraints and control action are discretized over the same time grid. A boundary value problem (BVP) is solved for each interval and additional continuity constraints are imposed. Due to the nature of the system dynamics and the imposed constraints, the optimization problem becomes a nonlinear program (NLP). This NLP is solved using Sequential Quadratic Programming (SQP) technique where the Quadratic Programs (QP) are solved by active set method using the qpOASES solver [7].
Note that, in case of infeasibility of the underlying QP, $\ell 1$ penalized slack variables are introduced to relax all constraints.
The controller is implemented in a receding horizon fashion where only the first computed control action is applied to the system, and the rest of the predicted state and control trajectory is used as initial guess for the Optimal Control Problem (OCP) to solve in the next iteration.

2.4 Linear Robust Model Predictive Control

Despite the robustness properties of the nominal MPC formulation, specific robust control variations of it exist when further robustness guarantees are required. The problem of linear robust model predictive control (RMPC) may be formulated as a Minimax optimization problem that is solved explicitly. As an optimality metric we may select the Minimum Peak Performance Measure (MPPM) for its known robustness properties. Figure 2 outlines the relevant building blocks [8].

Within this RMPC approach, the following linear time invariant representation of the system dynamics may be considered:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{G}\mathbf{w}_k \\ \mathbf{y}_{k+1} &= \mathbf{C}\mathbf{x}_k \end{aligned} \tag{8}$$

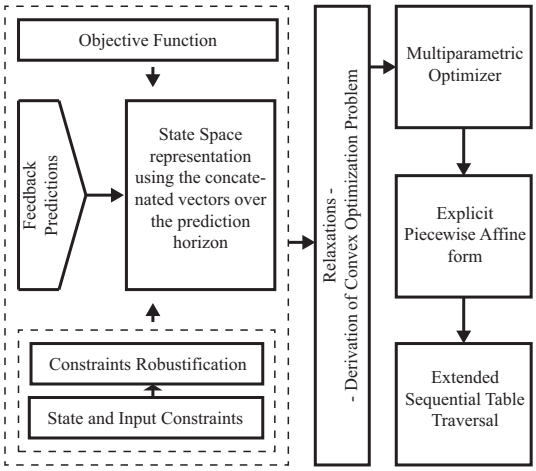


Fig. 2 Overview of the explicit RMPC optimization problem functional components.

where $\mathbf{x}_k \in \mathbf{X}, \mathbf{u}_k \in \mathbf{U}$ and the disturbing signals $\mathbf{w}_k$ are unknown but bounded ($\mathbf{w}_k \in \mathbf{W}$). Within this paper, box–constrained disturbances are considered ($\mathbf{W}_\infty = \{\mathbf{w} : ||\mathbf{w}||_\infty \leq 1\}$). Consequently, the RMPC problem will be formulated for the system representation and additive disturbance presented above. Let the following denote the concatenated versions of the predicted output, states, inputs and disturbances, where $[k+i|k]$ marks the values profile at time $k+i$, from time k.

$$\mathcal{Y} = \begin{pmatrix} \mathbf{y}_{k|k}^T & \mathbf{y}_{k+1|k}^T & \cdots & \mathbf{y}_{k+N-1|k}^T \end{pmatrix} \tag{9}$$

$$\mathcal{X} = \begin{pmatrix} \mathbf{x}_{k|k}^T & \mathbf{x}_{k+1|k}^T & \cdots & \mathbf{x}_{k+N-1|k}^T \end{pmatrix} \tag{10}$$

$$\mathcal{U} = \begin{pmatrix} \mathbf{u}_{k|k}^T & \mathbf{u}_{k+1|k}^T & \cdots & \mathbf{u}_{k+N-1|k}^T \end{pmatrix} \tag{11}$$

$$\mathcal{W} = \begin{pmatrix} \mathbf{w}_{k|k}^T & \mathbf{w}_{k+1|k}^T & \cdots & \mathbf{w}_{k+N-1|k}^T \end{pmatrix} \tag{12}$$

where $\mathcal{X} \in \mathbb{X}^N = \mathbf{X} \times \mathbf{X} ... \times \mathbf{X}, \mathcal{U} \in \mathbb{U}^N = \mathbf{U} \times \mathbf{U} ... \times \mathbf{U}, \mathcal{W} \in \mathbb{W}^N = \mathbf{W} \times \mathbf{W} \times ... \times \mathbf{W}$. The predicted states and outputs present linear dependency on the current state, the future control input and the disturbance, and thus the following holds:

$$\begin{aligned} \mathcal{X} &= \mathcal{A}\mathbf{x}_{k|k} + \mathcal{B}\mathcal{U} + \mathcal{G}\mathcal{W} \\ \mathcal{Y} &= \mathcal{C}\mathcal{X} \end{aligned} \tag{13}$$

where $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{G}$ are the stacked state vector matrices as in [8]. Subsequently, the RMPC problem based on the MPPM (MPPM–RMPC) may be formulated as:

$$\begin{aligned} \min_u \max_w \quad & ||\mathcal{Y}||_\infty, \; ||\mathcal{Y}||_\infty = \max_j ||\mathbf{y}_{k+j|k}||_\infty \\ s.t. \quad & \mathbf{u}_{k+j|k} \in \mathbf{U}, \; \forall \, \mathbf{w} \in \mathbf{W} \\ & \mathbf{x}_{k+j|k} \in \mathbf{X}, \; \forall \, \mathbf{w} \in \mathbf{W} \\ & \mathbf{w}_{k+j|k} \in \mathbf{W} \end{aligned} \tag{14}$$

2.4.1 Feedback Predictions

Following the aforementioned formulation, the optimization problem will tend to become conservative as the optimization essentially computes an open–loop control sequence. Feedback predictions is a method to encode the knowledge that a receding horizon approach is followed. Towards their incorporation, a type of feedback control structure has to be assumed. Among the feedback parametrizations that are known to lead to a convex problem space, the following is selected [9, 10]:

$$\mathcal{U} = \mathcal{L}\mathcal{W} + \mathcal{V}, \;\; \mathcal{V} = \begin{pmatrix} v_{k|k}^T & v_{k+1|k}^T & \cdots & v_{k+N-1|k}^T \end{pmatrix} \tag{15}$$

$$T\mathcal{L} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ L_{10} & 0 & 0 & \cdots & 0 \\ L_{20} & L_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{(N-1)0} & L_{(N-1)1} & \cdots & L_{(N-1)(N-2)} & 0 \end{pmatrix} \tag{16}$$

Employing this feedback predictions parameterization, the control sequence is now parameterized directly in the uncertainty, and the matrix $\mathcal{L}$ describes how the control action uses the disturbance vector. Inserting this parametrization yields the following representation, where $\mathcal{V}$ becomes now the RMPC–manipulated action:

$$\mathcal{X} = \mathcal{A}\mathbf{x}_{k|k} + \mathcal{B}\mathcal{V} + (\mathcal{G} + \mathcal{B}\mathcal{L})\mathcal{W} \tag{17}$$

$$\mathcal{U} = \mathcal{L}\mathcal{W} + \mathcal{V} \tag{18}$$

and the mapping from $\mathcal{L}, \mathcal{V}$ to $\mathcal{X}$,$\mathcal{U}$ is now bilinear. This allows the formulation of the minimax MPC as a convex optimization problem [9]. Furthermore, let:

$$\mathcal{F}_u = \begin{pmatrix} f_u^T & f_u^T & \cdots & f_u^T \end{pmatrix} \tag{19}$$

$$\mathcal{F}_x = \begin{pmatrix} f_x^T & f_x^T & \cdots & f_x^T \end{pmatrix} \tag{20}$$

denote the concatenated –over the prediction horizon– versions of the input and state constraints f_u and f_x. More specifically, $f_u = [f_u(1)^{\max}, f_u(1)^{\min}...]$ and $f_x = [f_x(1)^{\max}, f_x(1)^{\min}...]$ where $f_u(i)^{\max}, f_u(i)^{\min}$ represent the maximum and minimum allowed input values of the i–th input, while $f_x(j)^{\max}, f_x(j)^{\min}$ represent the maximum and minimum acceptable/safe state configurations of the j–th state.

$$\begin{aligned} \min_{\mathcal{V},\mathcal{L},\tau} \quad & \tau \\ s.t. \quad & ||\mathcal{C}(\mathcal{A}x_{k|k} + \mathcal{B}\mathcal{V} + (\mathcal{G} + \mathcal{B}\mathcal{L})\mathcal{W})||_\infty \leq \tau, \; \forall \, \mathcal{W} \in \mathbb{W}^N \\ & \mathcal{E}_u(\mathcal{V} + \mathcal{L}\mathcal{W}) \leq \mathcal{F}_u, \; \forall \, \mathcal{W} \in \mathbb{W}^N \\ & \mathcal{E}_x(\mathcal{A}x_{k|k} + \mathcal{B}\mathcal{V} + (\mathcal{G} + \mathcal{B}\mathcal{L})\mathcal{W}) \leq \mathcal{F}_x, \; \forall \, \mathcal{W} \in \mathbb{W}^N \\ \mathcal{E}_u = \; & \mathrm{diag}^N \mathbf{E}_u, \;\; \mathcal{E}_x = \mathrm{diag}^N \mathbf{E}_x, \;\; \tau > 0 \end{aligned} \tag{21}$$

within which: a) $\mathbf{E}_x$, $\mathbf{E}_u$ are matrices that allow the formulation of the state and input constraints in Linear Matrix Inequality (LMI) form, b) $\mathrm{diag}^N \Lambda_i$ is a block

diagonal matrix with Λ_i being the matrix that fills each diagonal block and allows the incorporation of the state and input constraints. Within this formulation τ is defined as a positive scalar value that bounds (from above) the objective function. The peak constraint may be equivalently reformulated as:

$$\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L})\mathcal{W} \leq \tau\mathbf{1},\ \forall\ \mathcal{W}\in\mathbb{W}^N \tag{22}$$

$$-\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})-\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L})\mathcal{W} \leq \tau\mathbf{1},\ \forall\ \mathcal{W}\in\mathbb{W}^N \tag{23}$$

where $\mathbf{1}$ is a vector of ones $(1\ 1\ \cdots\ 1)^T$ with suitable dimensions. Satisfaction of these uncertain inequalities is based on robust optimization methods.

2.4.2 Robust Uncertain Inequalities Satisfaction

Since box–constrained disturbances ($\mathbf{w}\in\mathbf{W}_\infty$) are assumed, the following holds:

$$\max_{|x|\leq 1} c^T x = ||c||_1 = |c^T|\mathbf{1} \tag{24}$$

This equation holds as $\max_{|x|\leq 1} c^T x = \max_{|x|\leq 1}\sum c_i x_i = \sum c_i \text{sign}(c_i) = ||c||_1$. Consequently, the uncertain constraints with $w\in\mathbb{W}_\infty$ are satisfied as long as [9]:

$$\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+|\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L})|\mathbf{1} \leq \tau\mathbf{1} \tag{25}$$

$$-\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+|\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L})|\mathbf{1} \leq \tau\mathbf{1} \tag{26}$$

To handle these constraints in a linear programming fashion [5], the term $|\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L})|$ is bounded from above by introducing a matrix variable $\Gamma\succ 0$:

$$\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L}) \leq \Gamma \tag{27}$$

$$-\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L}) \leq \Gamma \tag{28}$$

and the peak constraint is guaranteed as long as:

$$\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+\Gamma\mathbf{1} \leq \tau\mathbf{1} \tag{29}$$

$$-\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+\Gamma\mathbf{1} \leq \tau\mathbf{1} \tag{30}$$

2.4.3 Robust State and Input Constraints

To robustly satisfy hard constraints on the input and the states along the prediction horizon, a new matrix $\Omega\succ 0$ is introduced and the constraints are reformulated as:

$$\begin{pmatrix}\mathcal{E}_x(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})\\ \mathcal{E}_u\mathcal{V}\end{pmatrix}+\Omega\mathbf{1} \leq \begin{pmatrix}\mathcal{F}_x\\ \mathcal{F}_u\end{pmatrix} \tag{31}$$

$$\begin{pmatrix}\mathcal{E}_x(\mathcal{G}+\mathcal{B}\mathcal{L})\\ \mathcal{E}_u\mathcal{L}\end{pmatrix} \leq \Omega \tag{32}$$

$$-\begin{pmatrix}\mathcal{E}_x(\mathcal{G}+\mathcal{B}\mathcal{L})\\ \mathcal{E}_u\mathcal{L}\end{pmatrix} \leq \Omega \tag{33}$$

Optimizing the control sequence, while robustly satisfying the state and input constraints is of essential importance for the flight control of micro aerial vehicles.

2.4.4 Minimum Peak Performance Robust MPC formulation

Based on the aforementioned derivations, the total MPPM–RMPC formulation is solved subject to element–wise bounded disturbances and feedback predictions through the following linear programming problem:

$$\min_{\mathcal{V},\mathcal{L},\tau,\Omega,\Gamma} \quad \tau \tag{34}$$

$$\begin{aligned} s.t. \quad & \mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+\Gamma\mathbf{1} \leq \tau\mathbf{1} \\ & -\mathcal{C}(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V})+\Gamma\mathbf{1} \leq \tau\mathbf{1} \\ & \mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L}) \leq \Gamma \\ & -\mathcal{C}(\mathcal{G}+\mathcal{B}\mathcal{L}) \leq \Gamma \end{aligned}$$

$$\begin{aligned} & \mathcal{E}_x(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V}) \\ & \begin{pmatrix} \mathcal{E}_x(\mathcal{A}x_{k|k}+\mathcal{B}\mathcal{V}) \\ \mathcal{E}_u\mathcal{V} \end{pmatrix} + \Omega\mathbf{1} \leq \begin{pmatrix} \mathcal{F}_x \\ \mathcal{F}_u \end{pmatrix} \\ & \begin{pmatrix} \mathcal{E}_x(\mathcal{G}+\mathcal{B}\mathcal{L}) \\ \mathcal{E}_u\mathcal{L} \end{pmatrix} \leq \Omega \\ & -\begin{pmatrix} \mathcal{E}_x(\mathcal{G}+\mathcal{B}\mathcal{L}) \\ \mathcal{E}_u\mathcal{L} \end{pmatrix} \leq \Omega \end{aligned} \tag{35}$$

2.4.5 Multiparametric Explicit Solution

The presented RMPC strategy requires the solution of a linear programming problem. However, a multiparametric–explicit solution is possible due to the fact that the control action takes the general form [6]:

$$\mathbf{u}_k = \mathbf{F}^r\mathbf{x}_k + \mathbf{Z}^r, \text{ if } \mathbf{x}_k \in \Pi^r \tag{36}$$

where Π^i, $r = 1,...,N^r$ are the regions of the receding horizon controller. The r–th control law is valid if the state vector $\mathbf{x}_k$ is contained in a convex polyhedral region $\Pi^r = \{\mathbf{x}_k \,|\mathbf{H}^r\mathbf{x}_k \leq \mathbf{K}^r\}$ computed and described in h–representation [11]. Such a fact enables fast real–time execution even in microcontrollers with very limited computing power. In this framework, the real–time code described in [8] is employed.

3 Model-based Trajectory Tracking Controller for Multi-rotor System

In this section, we present a simplified model of multi-rotor system that can be used for model-based control to achieve trajectory tracking, and we present a linear and nonlinear model predictive controller for trajectory tracking.

3.1 Multirotor System Model

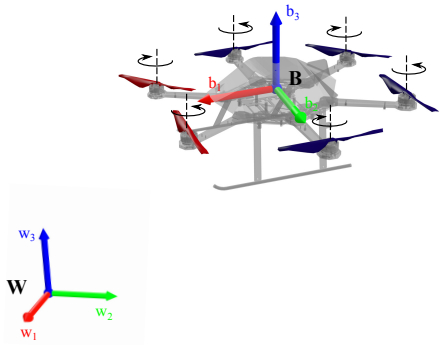


Fig. 3 Illustration of the Firefly hexacopter from Ascending Technologies with attached body fixed frame $\mathbb{B}$ and inertial frame $\mathbb{W}$.

The 6DoF pose of the multi-rotor system can be defined by assigning a fixed inertial frame $\mathbb{W}$ and body frame $\mathbb{B}$ attached to the vehicle as shown in Figure 3. We denote by p the position of the origin of frame $\mathbb{B}$ in frame $\mathbb{W}$ expressed in frame $\mathbb{W}$, by R the rotation matrix of frame $\mathbb{B}$ in frame $\mathbb{W}$ expressed in frame $\mathbb{W}$. Moreover, we denote by ϕ, θ and ψ the roll, pitch and yaw angles of the vehicle. In this model we assume a low level attitude controller that is able to track desired roll and pitch ϕ_d, θ_d angles with a first order behavior. The first order inner-loop approximation provides sufficient information to the MPC to take into account the low level controller behavior. The inner-loop first order parameters can be identified through classic system identification techniques. The non-linear model used for trajectory tracking of multi-rotor system is shown in Equations (37).

$$\begin{aligned}
\dot{p}(t) &= v(t) \\
\dot{v}(t) &= R(\psi,\theta,\phi)\begin{pmatrix}0\\0\\T\end{pmatrix} + \begin{pmatrix}0\\0\\-g\end{pmatrix} - \begin{pmatrix}A_x & 0 & 0\\0 & A_y & 0\\0 & 0 & A_z\end{pmatrix} v(t) + d(t) \\
\dot{\phi}(t) &= \frac{1}{\tau_\phi}\left(K_\phi \phi_d(t) - \phi(t)\right) \\
\dot{\theta}(t) &= \frac{1}{\tau_\theta}\left(K_\theta \theta_d(t) - \theta(t)\right)
\end{aligned} \tag{37}$$

where v indicates the vehicle velocity, g is the gravitational acceleration, T is the mass normalized thrust, A_x, A_y, A_z indicate the mass normalized drag coefficients, d is external disturbance. τ_ϕ, K_ϕ and τ_θ, K_θ are the time constant and gain of inner-loop behavior for roll angle and pitch angle respectively.

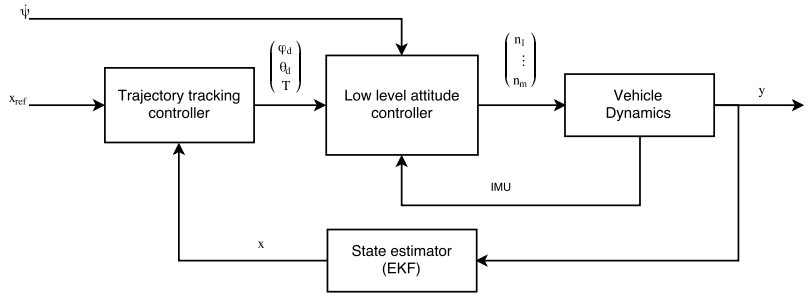


Fig. 4 Controller scheme for multi-rotor system. $n_1 \dots n_m$ are the $i-th$ rotor speed and y is the measured vehicle state.

The cascade controller structure assumed in this chapter is shown in Figure 4.

3.2 Linear MPC

In this subsection we show how to formulate a linear MPC to achieve trajectory tracking for multi-rotor system and integrate it into ROS. The optimization problem presented in Equation (3) is solved by generating a C-code solver using the CVXGEN framework [12]. CVXGEN generates a high speed solver for convex optimization problems by exploiting the problem structure. For clarity purposes, we rewrite the optimization problem here and show how to generate a custom solver using CVXGEN. The optimization problem is given by

$$\begin{aligned}
&\min_{U,X} \left(\sum_{k=0}^{N-1} \left(x_k - x_{ref,k}\right)^T Q_x \left(x_k - x_{ref,k}\right) + \left(u_k - u_{ref,k}\right)^T R_u \left(u_k - u_{ref,k}\right) + \left(u_k - u_{k-1}\right)^T R_\Delta \left(u_k - u_{k-1}\right) \right) \\
&\quad + \left(x_N - x_{ref,N}\right)^T P \left(x_N - x_{ref,N}\right) \\
&\quad \text{subject to} \quad x_{k+1} = Ax_k + Bu_k + B_d d_k; \\
&\qquad\qquad\quad\;\; d_{k+1} = d_k, \quad k = 0, \ldots, N-1 \\
&\qquad\qquad\quad\;\; u_k \in \mathbb{U} \\
&\qquad\qquad\quad\;\; x_0 = x(t_0), \quad d_0 = d(t_0).
\end{aligned} \tag{38}$$

To generate a solver for the aforementioned optimization problem, the following problem description is used in CVXGEN.

```
dimensions
  m = 3    # dimension of inputs.
  nd = 3   # dimension of disturbances.
  nx = 8   # dimension of state vector.
  T = 18   # horizon - 1.
end

parameters
  A (nx,nx)      # dynamics matrix.
  B (nx,m)       # transfer matrix.
  Bd (nx,nd)       # disturbance transfer matrix
  Q_x (nx,nx) psd # state cost, positive semidifined.
  P (nx,nx) psd # final state penalty, positive semidifined.
  R_u (m,m) psd # input penalty, positive semidifined.
  R_delta (m,m) psd # delta input penalty, positive semidifined.
  x[0] (nx)      # initial state.
  d    (nd)      # disturbances.
  u_prev (m)       # previous input applied to the system.
  u_max (m)      # input amplitude limit.
  u_min (m)      # input amplitude limit.
  x_ss[t] (nx), t=0..T+1     # reference state.
  u_ss[t] (m), t=0..T          # reference input.
end

variables
  x[t] (nx), t=1..T+1    # state.
  u[t] (m),  t=0..T      # input.
end

minimize
```

```
31   quad(x[0]-x_ss[0], Q_x) + quad(u[0]-u_ss[0], R_u) + quad(u[0] -
       u_prev, R_delta) + sum[t=1..T](quad(x[t]-x_ss[t], Q) + quad(
     u[t]-u_ss[t], R_u) + quad(u[t] - u[t-1], R_delta))+quad(x[T
     +1]-x_ss[T+1], P)
32 subject to
33   x[t+1] == A*x[t] + B*u[t] + Bd*d , t=0..T  # dynamics
34   u_min  <= u[t] <= u_max, t=0..T   # input constraint.
35 end
```

Before we show the integration of the generated solver into ROS, we discuss how to estimate the attitude loop parameters $\tau_\phi, K_\phi, \tau_\theta, K_\theta$ and the derivation of the discrete system model A, B, B_d.

3.2.1 Attitude Loop Parameters Identification

The attitude loop identification is a recommended process when no knowledge is available about the attitude controller used onboard of the vehicle. This is the case for many commercially available platforms. To perform this system identification, typically a pilot excites the vehicles axes in free flight. Attitude command and actual vehicle attitude (estimated using motion capture system if available or IMU) are logged with accurate time stamp. Typically two datasets are collected, one is used for parameters estimation and the other dataset is used for validation purpose. We will not go into details of system identification, interested readers can find more details in [13]. A set of scripts for attitude dynamics identification will be made open source upon acceptance of the chapter.

To perform parameters identification using the available scripts please follow the following steps:

1. Prepare the system and make sure you are able to log time stamped attitude commands and actual vehicle attitude.
2. Perform a flight logging the commands and vehicle attitude in a bag file. During the flight excite as much as possible each axis of the vehicle.
3. Repeat the flight test to collect validation dataset.
4. Set the correct bag files name and topics name in the provided script and run it.
5. The controller parameters will be displayed on screen with validation percentage to confirm the validity of the identification.

3.2.2 Linearization, Decoupling and Discretization

For controller design, the vehicle dynamics can be approximated around hovering condition where small attitude angles are assumed and vehicle heading aligned with inertial frame x axis (i.e. $\psi = 0$). The linearized system around hovering condition can be written as

$$\dot{x}(t) = \underbrace{\begin{pmatrix} 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&-A_x&0&0&g&0\\ 0&0&0&0&-A_y&0&0&-g\\ 0&0&0&0&0&-A_z&0&0\\ 0&0&0&0&0&0&-\frac{1}{\tau_\phi}&0\\ 0&0&0&0&0&0&0&-\frac{1}{\tau_\theta} \end{pmatrix}}_{A_c} x(t) + \underbrace{\begin{pmatrix} 0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&0\\ 0&0&1\\ \frac{K_\phi}{\tau_\phi}&0&0\\ 0&\frac{K_\theta}{\tau_\theta}&0 \end{pmatrix}}_{B_c} u(t) + \underbrace{\begin{pmatrix} 0&0&0\\ 0&0&0\\ 0&0&0\\ 1&0&0\\ 0&1&0\\ 0&0&1\\ 0&0&0\\ 0&0&0 \end{pmatrix}}_{B_{d,c}} d(t), \tag{39}$$

where the state vector is $x = \left[p^T, v^T, {}^{\mathbb{W}}\phi, {}^{\mathbb{W}}\theta\right]^T$, the input vector $u = \left[{}^{\mathbb{W}}\phi_d, {}^{\mathbb{W}}\theta_d, T\right]^T$ and the disturbance vector $d = [d_x, d_y, d_z]^T$. The c subscription indicate that this is a continuous time model. Note that in this linearization we marked the attitude to be in inertial frame $\mathbb{W}$ to get rid of the yaw angle ψ from the model. The attitude control action ${}^{\mathbb{W}}\phi_d, {}^{\mathbb{W}}\theta_d$ is computed in inertial frame and then converted to body frame by performing a rotation around z axis. The control action in the vehicle body frame $\mathbb{B}$ is given by

$$\begin{pmatrix} \phi_d \\ \theta_d \end{pmatrix} = \begin{pmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} {}^{\mathbb{W}}\phi_d \\ {}^{\mathbb{W}}\theta_d \end{pmatrix}. \tag{40}$$

After the computation of the control input, it is recommended to add a feed-forward term to compensate for coupling effects and to achieve better tracking performance. To compensate for vehicle non-zero attitude effects on thrust, and to achieve better tracking of dynamic trajectory, the following compensation scheme is employed.

$$\begin{aligned} \tilde{T} &= \frac{T+g}{\cos\phi\cos\theta} + {}^{\mathbb{B}}\ddot{z}_d \\ \tilde{\phi}_d &= \frac{g\phi_d - {}^{\mathbb{B}}\ddot{y}_d}{\tilde{T}} \\ \tilde{\theta}_d &= \frac{g\theta_d + {}^{\mathbb{B}}\ddot{x}_d}{\tilde{T}} \end{aligned} \tag{41}$$

where ${}^{\mathbb{B}}\ddot{x}_d, {}^{\mathbb{B}}\ddot{y}_d, {}^{\mathbb{B}}\ddot{z}_d$ are the desired trajectory acceleration expressed in vehicle body frame and quantities with tilde sign are the actual applied control input .
Given that the controller is implemented in discrete time, it is necessary to discretize the system dynamics (39). This can be done as follows

$$\begin{aligned} A &= e^{A_c T_s} \\ B &= \int_0^{T_s} e^{A_c \tau} d\tau B_c \\ B_d &= \int_0^{T_s} e^{A_c \tau} d\tau B_{d,c} \end{aligned} \tag{42}$$

where T_s is the sampling time. The computation of the terminal cost P matrix is done by solving the Algebraic Riccati Equation iteratively.

3.2.3 ROS Integration

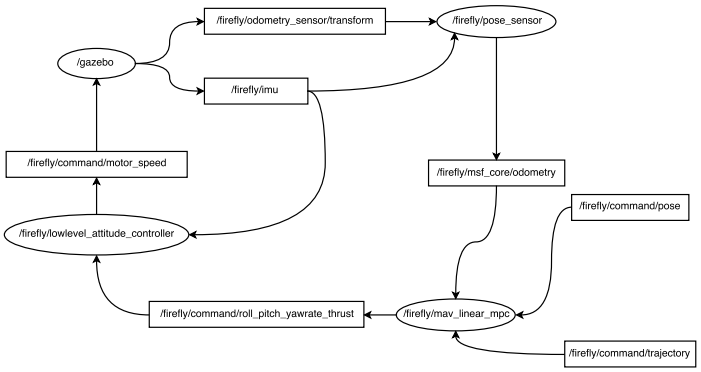


Fig. 5 ROS graph diagram showing various nodes and topics to control multi-rotor system.

The strategy followed for the ROS integration of the solver is to create a ROS node to interface the controller to ROS environment, while the controller, estimator and other components are implemented as C++ shared libraries that get linked to the node at compilation time. The controller node expects the vehicle state as a nav `odometry` message and publishes a custom message of type `RollPitchYawRateThrust` command message. The desired trajectory can be sent to the controller over a topic of type `MultiDOFJointTrajectory` or as single desired point over a topic of type `PoseStamped`. The advantage of passing the whole desired trajectory over single point is that the MPC can take into consideration the future desired trajectory and react accordingly. Figure 5 gives an overview of nodes and topics communication through a ros graph diagram.
Each time the controller receives a new odometry message, a control action is computed and published. Therefore it is important to guarantee that the state estimator is publishing an odometry message at the desired rate of control. We recommend to use this controller with the Modular Framework for MultiSensor Fusion [14]. An important point to consider when implementing such a controller is to reduce as much as possible the delays in the loop. A hint to minimize communication delay is to use the ROS transportation hints by passing `ros::TransportHints().tcpNoDelay()` flag to the odometry subscriber. The controller node publishes the following information:

1. Current desired reference as `tf`.
2. Desired trajectory as rviz marker.

3. Predicted vehicle state as rviz marker.

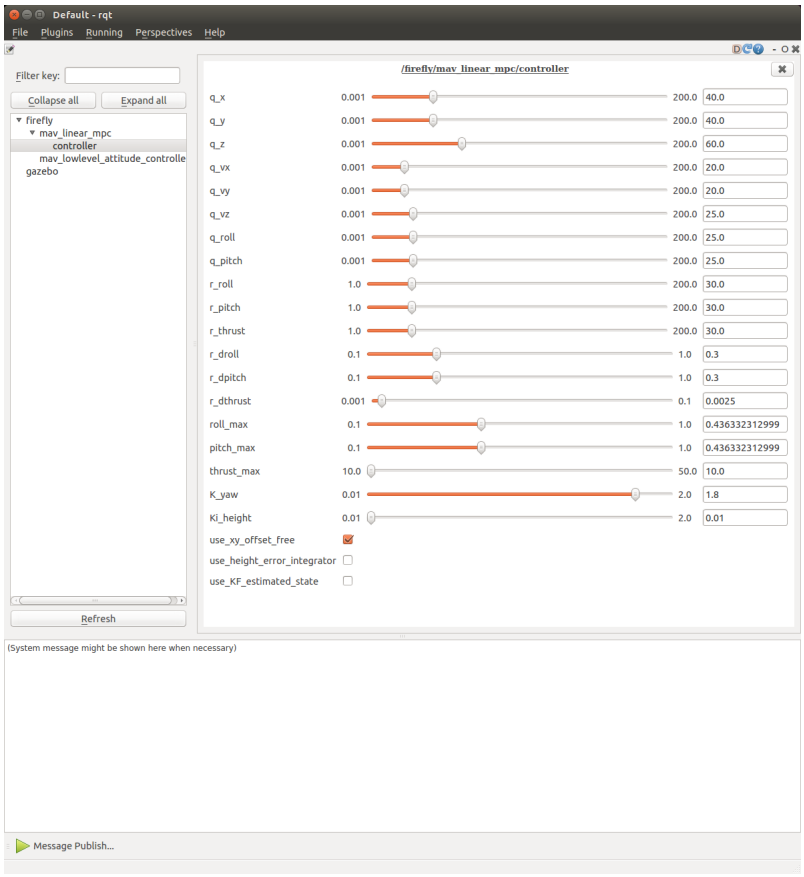


Fig. 6 Through dynamic reconfiguration it is possible to change the controller parameters online for tuning purposes.

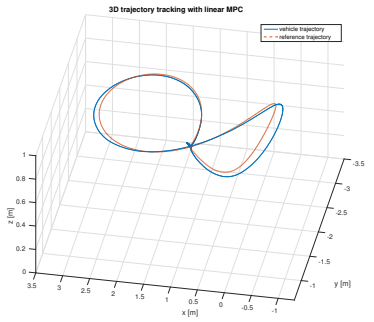
The controller parameters can be easily changed in run time from the dynamic reconfigure as shown in Figure 6.

An open source implementation of the presented controller can be found in `https://github.com/ethz-asl/mav_control_rw`

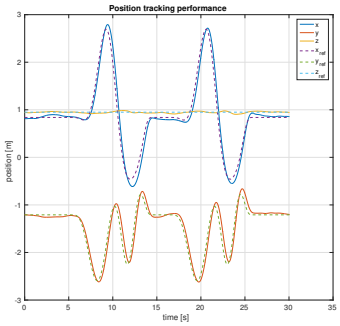
3.2.4 Experimental Results

To validate the controller performance, we track an aggressive trajectory. The controller is running onboard a Firefly hexacopter from Ascending Technologies with a NUC i7 computer onboard. An external motion capture system Vicon [15] is used

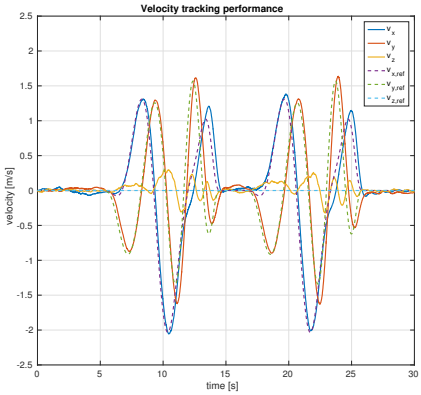
as pose sensor and fused with onboard inertial measurement unit (IMU) using the Multisensor Fusion Framework (MSF). The controller is running at 100 Hz and the prediction horizon is chosen to be 20 steps.



(a) Aggressive trajectory tracking using linear MPC. 3D trajectory plot.



(b) Position tracking performance using linear MPC



(c) Velocity tracking performance using linear MPC

Fig. 7 Aggressive trajectory tracking performance using linear MPC controller running onboard of Firefly hexacopter from Ascending Technologies.

Figure 7 shows the tracking performance of the controller.

3.3 Nonlinear MPC

In this subsection we use the full vehicle nonlinear model to design a continuous time nonlinear model predictive controller. The toolkit employed to generate the nonlinear solver is ACADO [16] which is able to generate very fast custom C code solvers for general optimal control problems OCP. The optimization problem can be written as

$$\begin{aligned} \min_{U,X} \int_{t=0}^{T} & \left(x(t)-x_{ref}(t)\right)^T Q_x \left(x(t)-x_{ref}(t)\right) + \left(u(t)-u_{ref}(t)\right)^T R_u \left(u(t)-u_{ref}(t)\right) dt \\ & + \left(x(T)-x_{ref}(T)\right)^T P \left(x(T)-x_{ref}(T)\right) \\ \text{subject to} \quad & \dot{x} = f(x,u); \\ & u(t) \in \mathbb{U} \\ & x(0) = x(t_0). \end{aligned} \tag{43}$$

To generate a solver for the aforementioned optimization problem, the following problem description is used in ACADO's Matlab interface.

```
clc;
clear all;
close all;

Ts = 0.1; %sampling time
EXPORT = 1;

DifferentialState position(3) velocity(3) roll pitch yaw;
Control roll_ref pitch_ref thrust;

%online data represent data that can be passed to the solver
    online.
OnlineData roll_tau;
OnlineData roll_gain;
OnlineData pitch_tau;
OnlineData pitch_gain;
OnlineData yaw_rate_command;
OnlineData drag_coefficient(3);
OnlineData external_disturbances(3);

n_XD = length(diffStates);
n_U = length(controls);

g = [0;0;9.8066];

%% Differential Equation

%define vehicle body z-axis expressed in inertial frame.
z_axis =  [(cos(yaw)*sin(pitch)*cos(roll)+sin(yaw)*sin(roll));...
           (sin(yaw)*sin(pitch)*cos(roll)-cos(yaw)*sin(roll));...
           cos(pitch)*cos(roll)];

```

```
droll = (1/roll_tau)*(roll_gain*roll_ref - roll);
dpitch = (1/pitch_tau)*(pitch_gain*pitch_ref - pitch);

f = dot([position, velocity; roll; pitch; yaw]) == ...
  [velocity ...;
  z_axis*thrust - g - diag(drag_coefficient)*velocity +
    external_disturbances;...
  droll;...
  dpitch;...
  yaw_rate_command];

h = [position; velocity; roll; pitch; roll_ref; pitch_ref; z_axis
    (3)*thrust-g(3)];

hN = [position; velocity];


%% NMPCexport
acadoSet('problemname', 'nmpc_trajectory_tracking');

N = 20;
ocp = acado.OCP( 0.0, N*Ts, N );

W_mat = eye(length(h));
WN_mat = eye(length(hN));
W = acado.BMatrix(W_mat);
WN = acado.BMatrix(WN_mat);

ocp.minimizeLSQ( W, h );
ocp.minimizeLSQEndTerm( WN, hN );
ocp.subjectTo(-deg2rad(45) <= [roll_ref; pitch_ref] <= deg2rad
    (45));
ocp.subjectTo( g(3)/2.0 <= thrust <= g(3)*1.5);
ocp.setModel(f);

mpc = acado.OCPexport( ocp );
mpc.set( 'HESSIAN_APPROXIMATION',          'GAUSS_NEWTON'        );
mpc.set( 'DISCRETIZATION_TYPE',            'MULTIPLE_SHOOTING'   );
mpc.set( 'SPARSE_QP_SOLUTION',           'FULL_CONDENSING_N2'    );
mpc.set( 'INTEGRATOR_TYPE',                'INT_IRK_GL4'         );
mpc.set( 'NUM_INTEGRATOR_STEPS',           N                     );
mpc.set( 'QP_SOLVER',                      'QP_QPOASES'          );
mpc.set( 'HOTSTART_QP',                    'NO'                  );
mpc.set( 'LEVENBERG_MARQUARDT',            1e-10                 );
mpc.set( 'LINEAR_ALGEBRA_SOLVER',          'GAUSS_LU'            );
mpc.set( 'IMPLICIT_INTEGRATOR_NUM_ITS',    4                     );
mpc.set( 'CG_USE_OPENMP',                  'YES'                 );
mpc.set( 'CG_HARDCODE_CONSTRAINT_VALUES', 'NO'                   );
mpc.set( 'CG_USE_VARIABLE_WEIGHTING_MATRIX', 'NO'                );


if EXPORT
  mpc.exportCode( 'mav_NMPC_trajectory_tracking' );
```

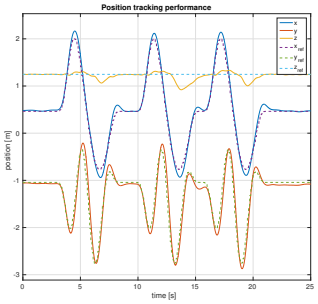
```
83
84    cd mav_NMPC_trajectory_tracking
85    make_acado_solver('../mav_NMPC_trajectory_tracking')
86    cd ..
87  end
```

3.3.1 ROS Integration

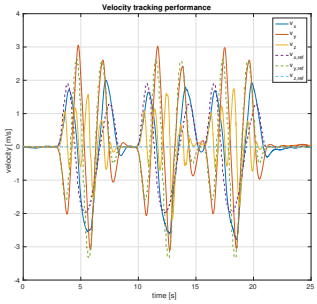
The ROS integration of the nonlinear controller follows the same guidelines of the linear MPC previously presented. An open source implementation of the previously presented controller can be found here. `https://github.com/ethz-asl/mav_control_rw`.

3.3.2 Experimental Results

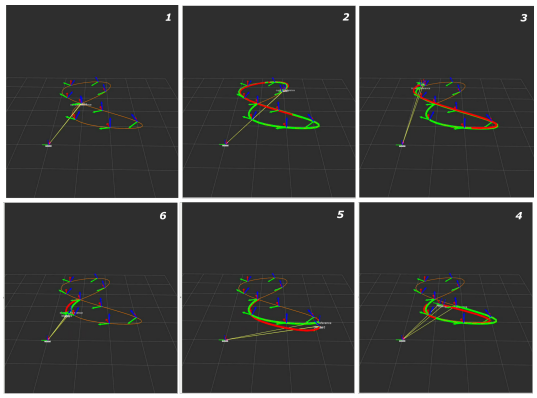
The same setup used for the linear MPC is used to evaluate the nonlinear MPC. The only difference is that we are currently evaluating the controller on a more aggressive trajectory as shown in Figure 8.



(a) Position tracking performance using nonlinear MPC



(b) Velocity tracking performance using nonlinear MPC



(c) Sequence of rviz screen shot showing the reference trajectory in green and predicted vehicle state in red.

Fig. 8 Aggressive trajectory tracking performance using nonlinear MPC controller running onboard of Firefly hexacopter from Ascending Technologies.

3.3.3 Robust Linear Model Predictive Control for Multirotor System

In order to evaluate the proposed RMPC, a set of test–cases were conducted using the structurally modified AscTec Hummingbird quadrotor (ASLquad) shown in Figure 9. For the implementation of the RMPC, a software framework around ROS was developed. In particular, a set of low–level drivers and nodes handle the communication with the attitude and motor control on–board the aerial robot and therefore enabling us to provide attitude and thrust references through the RMPC. MATLAB was used to derive and compute the explicity formulation of the RMPC, while the complete explicit algorithm overviewed in Section 2.4.5 was implemented within a SIMULINK block. Auto–code generation was then employed to extract the C–code equivalent of this controller, which was then wrapped around a ROS node and integrated into the overall software framework. For the described experiments with the RMPC, full state feedback is provided through external motion capture, while an alignment step to account for relative orientation of the on–board attitude and heading estimation system also takes place.

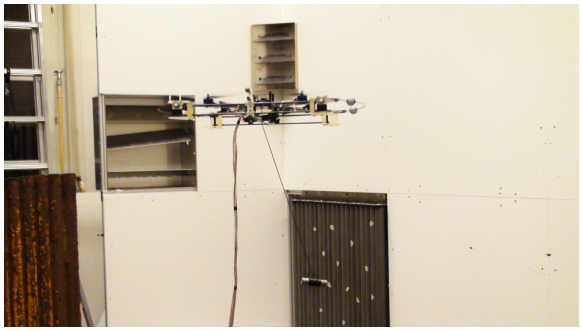


Fig. 9 Instance of an RMPC test for slung load operations.

Below we present the results on a) trajectory tracking subject to wind disturbances, and b) slung load operations, while further results are available at [8]. For the presented experiments, the sampling time was set to $T_s = 0.08$s, the prediction horizon was set to $N = 6$ for both of them, while the following state and input constraints were considered:

$$\begin{bmatrix} \mathbf{I}_{7\times7} & \mathbf{0}_{7\times7} \\ \mathbf{0}_{7\times7} & -\mathbf{I}_{7\times7} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \theta \\ \phi \\ \theta^r \\ \phi^r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \theta \\ \phi \\ \theta^r \\ \phi^r \end{bmatrix} \leq \begin{bmatrix} 1.5\text{m/s} \\ 1.5\text{m/s} \\ 1.5\text{m/s} \\ \pi/4\text{rad} \\ \pi/4\text{rad} \\ \pi/4\text{rad} \\ \pi/4\text{rad} \\ 1.5\text{m/s} \\ 1.5\text{m/s} \\ 1/5\text{m/s} \\ \pi/4\text{rad} \\ \pi/4\text{rad} \\ \pi/4\text{rad} \\ \pi/4\text{rad} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{I}_{6\times6} & \mathbf{0}_{6\times6} \\ \mathbf{0}_{6\times6} & -\mathbf{I}_{6\times6} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \gamma^r \\ \phi^r \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \gamma^r \\ \phi^r \end{bmatrix} \leq \begin{bmatrix} 1.5\text{m/s} \\ 1.5\text{m/s} \\ 1.5\text{m/s} \\ \pi/4\text{rad} \\ \pi/6\text{rad} \\ \pi/4\text{rad} \\ 1.5\text{m/s} \\ 1.5\text{m/s} \\ 1/5\text{m/s} \\ \pi/4\text{rad} \\ \pi/6\text{rad} \\ \pi/4\text{rad} \end{bmatrix} \tag{44}$$

Regarding the first experimental test–case, an 80W electric fan was pointed to the ASLquad, while the RMPC was acting in order to track a helical path despite the turbulent wind disturbance. As can be observed in Figure 10, the tracking response remains precise and only a minor influence from the external disturbance is observed. Note that the box–constraints selection is sufficient to account –up to some extent– for the kind of disturbance as although the dynamics of the wind disturbance are unknown to the controller, its effect may be considered as bounded.

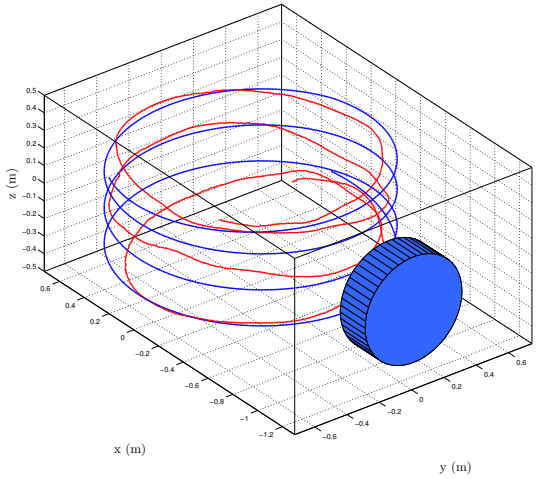


Fig. 10 Trajectory tracking subject to wind disturbances using the RMPC framework applied tothe ASLquad.

Consequently, the capabilities of performing slung load operations were considered. As shown in Figure 11 where a forcible external disturbance is also applied onto the slung load (a hit on the load), highly precise position hold and disturbance rejection results were achieved. This is despite the fact that the slung load introduces disturbances that are phase–delayed compared to the vehicle states and the controller is not augmented regardings its state to incorporate the load's motion. For

this experiment, a 0.16kg load was utilized and was attached through a compressible string with a length of 0.65m .

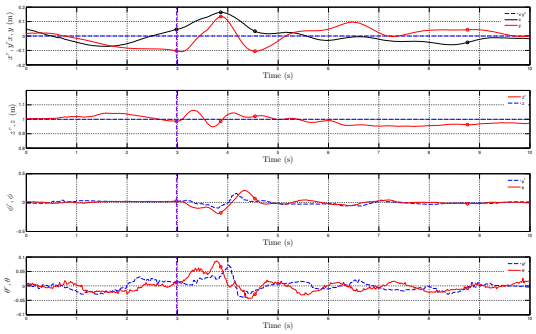


Fig. 11 RMPC performance against disturbance of the slung load which induces unpredicted disturbances on the vehicle. At time $t \approx 3$s a forcible external disturbance (hit) is applied on the load, however the RMPC manages to quickly and accurately stabilize the vehicle.

Once the capabilities of the proposed RMPC to handle slung load operations even while subjected to strong disturbances were verified, the problem of trajectory tracking during slung load operations (i.e.transportation and delivery of goods in a Search–and–Rescue scenario or as part of a product delivery mission), was examined. Figure 12 presents the results achieved when continuously tracking a square reference trajectory for 10 times while the same slung load is attached onto the quadrotor platform. As shown, efficient and robust results were derived, indicative of the capabilities of the proposed control scheme to effectively execute such operations.

In the last presented trajectory tracking experiments, a slow response was commanded. However, more agile maneuvers could also be considered even during slung load operations. To this purpose, a step reference of 1.25m was commanded with the velocity and attitude references being set to zero. The results are shown in Figure 13 and as depicted, a satisfactory response was derived although some overshoot is observed.

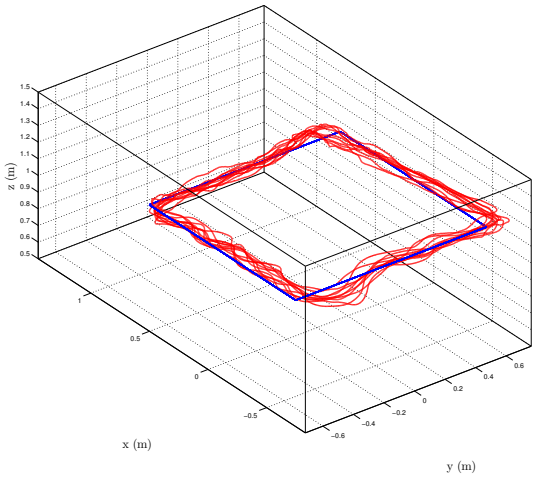


Fig. 12 Trajectory tracking during slung load operations using the ASLquad: the controller tracks the same square trajectory for 10 times with small deviations from the reference despite the significant and phase–delayed disturbances introduced from the load.

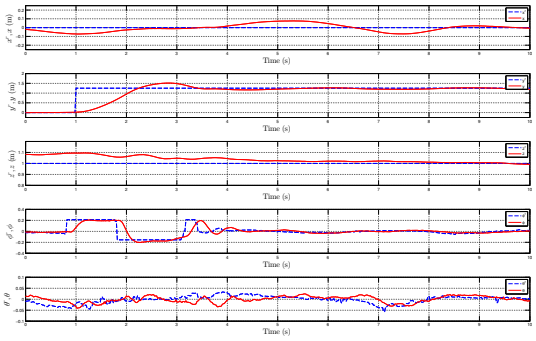


Fig. 13 Lateral step response during slung load operations: minimal overshoot is observed.

4 Model-based Trajectory Tracking Controller for Fixed-wing UAVs

In this section, we describe the modeling of simplified fixed-wing aircraft flight dynamics, closed-loop low-level system identification methodology, and control objective design necessary for a general-form high-level nonlinear model predictive trajectory tracking controller. As the vast majority of fixed-wing flight is conducted at fixed altitudes, we focus our presentation on planar lateral-directional position control; though, the techniques employed can easily be extended towards three-dimensional applications, assuming a suitably stabilizing longitudinal low-level controller.

4.1 Fixed-wing flight dynamics & identification

Lateral-directional dynamics for a fixed-wing system are defined in the intertial frame $\mathbb{I}$, in local coordinates composed of Northing $\hat{n}$ and Easting $\hat{e}$ components, and body frame $\mathbb{B}$. In reality, aerodynamic effects result in an additional "wind" frame, where the aircraft may slip, causing the airspeed vector $\mathbf{v}$ to devate from the body-$\hat{x}$-axis. In our simplified model, we will assume low-level control is designed such that slip is regulated appropriately, and we will assume wind and body frames are identical. It is worth also noting our planar assumption further entails the assumption that we fly at a fixed altitude. The dynamic equation are defined in Equation (45).

$$\begin{aligned} \dot{n} &= V\cos\psi + w_n \\ \dot{e} &= V\sin\psi + w_e \\ \dot{\psi} &= \tfrac{g\tan\phi}{V} \\ \dot{\phi} &= p \\ \dot{p} &= b_0\phi_r - a_1 p - a_0\phi \end{aligned} \tag{45}$$

where n and e are the Northing and Easting positions, respectively, ψ is the yaw angle, V is the air-mass-relative airspeed, ϕ is the roll angle, p is the roll rate, g is the acceleration of gravity, and w_n and w_e are the Northing and Easting components of the wind vector $\mathbf{w}$, respectively. Note that roll ϕ and yaw ψ angles are defined about the body frame $\mathbb{B}$. This distinction is important when considering flight dynamics in wind, where the ground-relative flight path of the vehicle is defined as the course angle χ from North $\hat{n}$ to the ground speed vector, $\mathbf{v}_g$, see Fig. 14. Additionally, note that we choose a second-order model with no zeros to describe the rolling dynamic with respect to roll references ϕ_r, where a_0, a_1, and b_0 are model coefficients. Higher order dynamics could be used, however we have found through the identification procedures outlined in Section 4.1.1, that second-order fits appropriately model the closed loop low-level autopilot attitude control response; further, each increase in

order in turn increases the dimension of the control optimization problem, increasing computational expense.

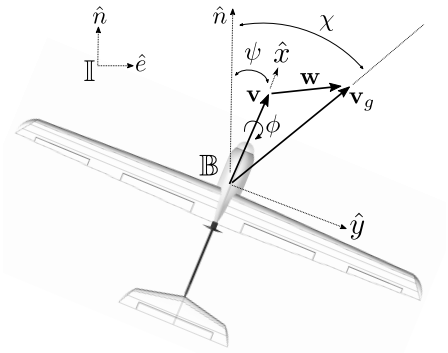


Fig. 14 Fixed-wing modeling definitions

4.1.1 Model identificaiton

Here, we will outline the basic methodology for closed loop model identification on fixed-wing aircraft. Towards these ends, it is assumed a low-level autopilot with onboard state estimation and attitude, airspeed, and altitude control functionality. Such capabilities are present in commerically available autopilot hardware and software such as the Pixhawk Autopliot, an open source/open hardware project started at ETH Zurich [1]. Tuning of the low-level loops will not be discussed, though these procedures are well documented in the literature as well as in practice on the Pixhawk website.

The control architecture shown in Figure 15 demonstrates a typical cascaded PID structure with attitude PI control and angular rate PD control. Additional compensation for slipping effects is considered for coordinated turns, i.e. a yaw damper signal, $r_r = \frac{g \sin \phi}{V}$. The TECS block indicates the use of Total Energy Control System for airspeed and altitude control, again fully implemented and documented on the pixhawk website. In three-dimensional extensions of the proposed lateral-directional NMPC, the high-level TECS block could be replaced.

The aim is to identify the closed loop low-level autpilot dynamic response when reacting to attitude commands. We will specifically discuss the roll dynamic here, however the same procedures discussed could be used for identification of pitching dynamics as well as airspeed, given that well-tuned low-level controllers are in place. Depending on the hardware used, autopilot source code could be modified for an identification option commanding repeatable excitation inputs, or in the case of utilizing a Linux operating system, a simple ROS node could be written to generate the same. For roll channel identification, pitch references to the low level controller

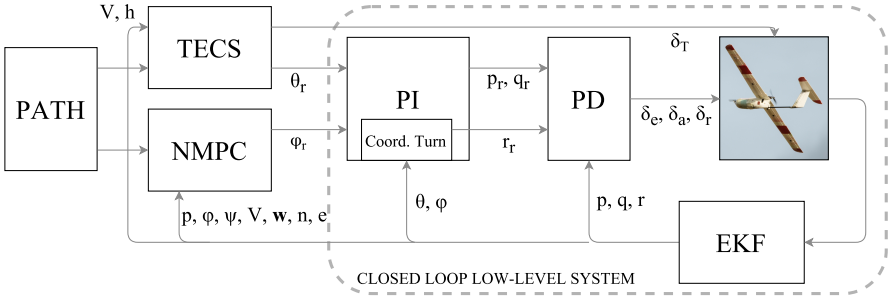


Fig. 15 Fixed-wing control architecture

should be held constant for holding altitude. Depending on the operational airspeed, the pitch reference may vary. 2-1-1 maneuvers, a modified doublet input consisting of alternating pulses with pulse widths in the ratio 2-1-1, are recommended, see Figure 16. Morelli[17] demonstrated that flight time required for the 2-1-1 maneuver is approximately one-sixth of the time required for the standard frequency sweep, enabling one to gather more data in the same flight time, which is often limited by battery capacity. At the same time, concatenated 2-1-1 maneuvers make for suitable identification inputs for both frequency and time domain system identification approaches, on par with frequency sweeps.

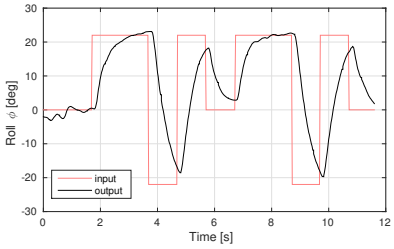


Fig. 16 Two concatenated 2-1-1 maneuvers from a flight experiment with a fixed-wing UAV.

It is good practice to perform all identification experiments on calm days with no wind and to persistently excite the control inputs. Data collection should consist of several 2-1-1 maneuvers for each of several setpoint magnitudes throughout the desired range. A similar set of data for both training and validation is required. To test the generalizability of the fit parameters, it is also worthwhile to include non-2-1-1 maneuvers in the validation set, e.g. arbitrary piloting (still within attitude control augmentation mode), to test the generalizability of the fit parameters. The authors provide a set of Matlab scripts that can perform the parameter identification

after data collection and format conversion. Further literature on closed loop system identification for fixed-wing vehicles can be found in [18] [17].

4.2 Nonlinear MPC

In this section, we formulate a Nonlinear MPC for general high-level fixed-wing lateral-directional trajectory tracking utilizing the model developed in the previous section. The generalized form involves augmentation of the vehicle model with trajectory information, including discretely defined sequential tracks. We use the ACADO Toolkit [16] for generation of a fast C code based nonlinear solver and integration scheme. The optimization problem (OCP) takes the continuous time form

$$\begin{aligned}
\min_{U,X} \int_{t=0}^{T} & \Big(\big(y(t)-y_{ref}(t)\big)^T Q_y \big(y(t)-y_{ref}(t)\big) + \big(u(t)-u_{ref}(t)\big)^T R_u \big(u(t)-u_{ref}(t)\big) \Big) dt \\
& + \big(y(T)-y_{ref}(T)\big)^T P \big(y(T)-y_{ref}(T)\big) \\
\text{subject to} \quad & \dot{x} = f(x,u) \\
& y = h(x,u) \\
& u(t) \in \mathbb{U} \\
& x(0) = x(t_0)\,.
\end{aligned} \tag{46}$$

The state vector is defined as $x = [n, e, \phi, \psi, p, x_{sw}]^T$, and control input $u = \phi_r$, where the augmented state x_{sw} is a switch state used within the horizon in the case that desired trajectories are piece-wise continuously, or generally discretely, defined. The switch variable has no dynamic until a switch condition is detected within the horizon, at which point an arbitrary differential α is applied for the remainder of the horizon calculations, see Equation (47). This assumes we only consider a maximum of two track segments in a given horizon, and ensures the optimization does not revert back to a previous track after switching within the horizon.

$$\dot{x}_{sw} = \begin{cases} \alpha & \text{switch condition met} \quad \| \quad x_{sw} > \text{threshold} \\ 0 & \text{else} \end{cases} \tag{47}$$

A general tracking objective is constructed for minimizing the position error to a given track,

$$e_t = (\mathbf{d} - \mathbf{p}) \times \bar{\mathbf{T}}_d \tag{48}$$

where $\bar{\mathbf{T}}_d$ is the unit tanget vector at the closest point $\mathbf{d}$ from the UAV position $\mathbf{p}$ to the current path $\mathscr{P}$, while also aligning the vehicle course with the desired trajectory direction, i.e. minimize

$$e_\chi = \chi_d - \chi \tag{49}$$

where $\chi_d = atan2(\bar{T}_{d_e}, \bar{T}_{d_n}) \in [-\pi, \pi]$. Here, we use the $atan2$ function from the standard C math library. See also Fig. 17. Use of this general objective formulation allows inputing any path shape, so long as the nearest point from the UAV position can be calculated and a direction of motion along the path is given for minimization throughout the horizon.

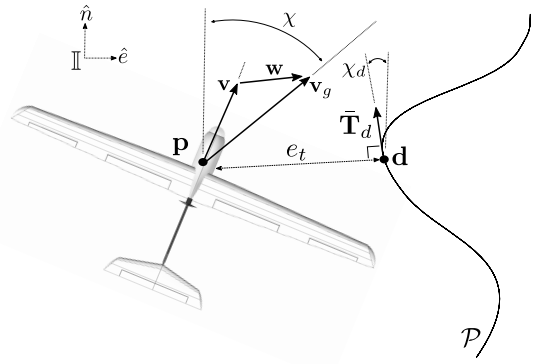


Fig. 17 Trajectory tracking objectives

A relevant example of switching trajectories is that of Dubins Car, or Dubins Aircraft in the three-dimensional case, path following, see [19]. Dubins paths can be used to describe the majority of desired flight maneuvers in a typical fixed-wing UAV mission. Further, using continuous curves such as arcs and lines allow time-invariant trajectory tracking, as oppose to desired positions in time, a useful quality when only spatial proximity to the track is desired and timing is less important; for instance, if energy conservation is required and a single low airspeed reference is given to be tracked. For the remainder of the section, we will consider Dubins segments as path inputs to the high-level controller, though it should be noted that the objective formulation is not limited to these.

Using the definitions in Equations (48) and (49), we formulate the objective vector $y = \left[e_t, e_\chi, \phi, p, \phi_r\right]^T$. We assume a fixed air-mass-relative airspeed V and two-dimensional wind vector $\mathbf{w}$, held constant throughout the horizon. These values are estimated and input to the optimization as online data. Also input as online data, the current and next sets of Dubins path parameters $\mathscr{P}_{cur}$, $\mathscr{P}_{next}$, where line parameters include $\mathscr{P} = \{type = 0, \mathbf{a}, \mathbf{b}\}$, $\mathbf{a}$ and $\mathbf{b}$ are two waypoints defining a straight segment, and arc parameters include $\mathscr{P} = \{type = 1, \mathbf{c}, R, dir, \psi_0, \Delta\psi\}$, $\mathbf{c}$ is the center point of the arc, R is the radius, dir is the loiter direction, and ψ_0, is the heading pointing towards the entrance point on the arc, and $\Delta\psi$ is the arclength traveled. The path segments are managed and switched based on an acceptance radius and heading direction criteria, see Figure 18.

All references are set to zero, except for the control input references. As the continuous time formulation does not allow slew rate penalties, it is possible to instead store the previous control horizon for input as weighted control references

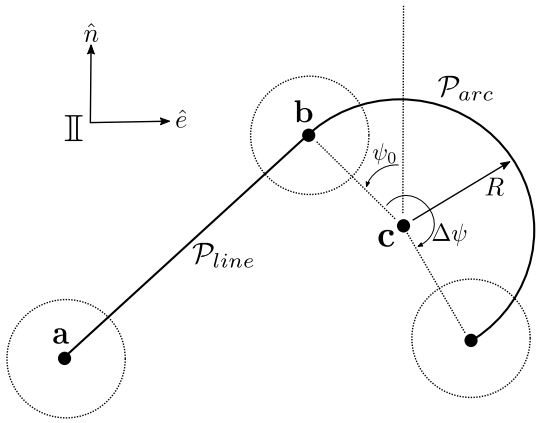


Fig. 18 Dubins path defintions

in the next NMPC iteration. This is done to avoid bang-bang control action when the NMPC iterations occur at relatively large steps, for fixed-wing we run the high-level controller at either 10 Hz or 20 Hz. As only the second step in the current control horizon is sent to the vehicle for tracking, the early horizon is penalized more heavily than the latter horizon values. This can be accomplished by weighting the horizon of controls with a decreasing quadratic function. More insights on the control formulation may be found in [20].

The following MATLAB script may be run to generate the ACADO solver in C code. Note when dealing with discontinuous functions in the model formulation, external models with accompanying external jacobians must be supplied, written in C. Further, if the objective function contains discontinuous functions, this may also be input externally. Here, we implement a numerical jacobian, though one could find individual analytic expressions for each discontinuous case and supply them to the code generation.

```
clc;
clear all;
close all;

Ts = 0.1; % model discretization
N  = 40;    % horizon length

% STATES ---------
DifferentialState n;            % (northing)                [m]
DifferentialState e;            % (easting)                 [m]
DifferentialState phi;          % (roll angle)              [rad]
DifferentialState psi;          % (yaw angle)               [rad]
DifferentialState p;            % (roll rate)               [rad/
    s]
DifferentialState x_sw;         % (switching state)         [~]

% CONTROLS -------
```

```
Control phi_r;                  % (roll angle reference)     [rad]

% ONLINE DATA ------
OnlineData V;                   % (airspeed)                 [m/s]
OnlineData pparam1;             % type=0  type=1
OnlineData pparam2;             % a_n       c_n
OnlineData pparam3;             % a_e       c_e
OnlineData pparam4;             % b_n       R
OnlineData pparam5;             % b_e       dir
OnlineData pparam6;             % --        psi0
OnlineData pparam7;             % --        dpsi
OnlineData pparam1_next;        % type=0    type=1
OnlineData pparam2_next;        % a_n       c_n
OnlineData pparam3_next;        % a_e       c_e
OnlineData pparam4_next;        % b_n       R
OnlineData pparam5_next;        % b_e       dir
OnlineData pparam6_next;        % --        psi0
OnlineData pparam7_next;        % --        dpsi
OnlineData wn;                  % (northing wind)            [m/s]
OnlineData we;                  % (easting wind)             [m/s]

% OPTIMAL CONTROL PROBLEM ------

% lengths
n_X = length(diffStates);       % states
n_U = length(controls);         % controls
n_Y = 4;                        % state/outputs objectives
n_Z = 1;                        % control objectives
n_OD = 17;                      % online data

% weights
Q = eye(n_Y+n_Z,n_Y+n_Z);
Q = acado.BMatrix(Q);

QN = eye(n_Y,n_Y);
QN = acado.BMatrix(QN);



% optimal control problem
acadoSet('problemname', 'nmpc');
ocp = acado.OCP( 0.0, N*Ts, N );

% minimization
ocp.minimizeLSQ( Q, 'evaluateLSQ' );
ocp.minimizeLSQEndTerm( QN, 'evaluateLSQEndTerm' );

% external model
ocp.setModel('model', 'rhs', 'rhs_jac');
ocp.setDimensions( n_X, n_U, n_OD, 0 );

ocp.subjectTo( -35*pi/180 <= phi_r <= 35*pi/180 );

setNOD(ocp, n_OD);
```

```
71
72 % export settings
73 nmpc = acado.OCPexport( ocp );
74 nmpc.set( 'HESSIAN_APPROXIMATION', 'GAUSS_NEWTON' );
75 nmpc.set( 'DISCRETIZATION_TYPE', 'MULTIPLE_SHOOTING' );
76 nmpc.set( 'SPARSE_QP_SOLUTION', 'FULL_CONDENSING' );
77 nmpc.set( 'INTEGRATOR_TYPE', 'INT_IRK_GL4' );
78 nmpc.set( 'NUM_INTEGRATOR_STEPS', N );
79 nmpc.set( 'QP_SOLVER', 'QP_QPOASES' );
80 nmpc.set( 'HOTSTART_QP', 'YES' );
81 nmpc.set( 'LEVENBERG_MARQUARDT', 1e-10 );
82 nmpc.set( 'GENERATE_MAKE_FILE', 'YES' );
83 nmpc.set( 'GENERATE_TEST_FILE', 'YES' );
84 nmpc.set( 'GENERATE_SIMULINK_INTERFACE', 'YES' );
85 nmpc.set( 'CG_HARDCODE_CONSTRAINT_VALUES', 'YES' );
86 nmpc.set( 'CG_USE_VARIABLE_WEIGHTING_MATRIX', 'YES' );
87
88 % export
89 copyfile('../acado/external_packages/qpoases', ...
90     'export_nmpc/qpoases')
91 nmpc.exportCode( 'export_nmpc' );
92
93 cd export_nmpc
94 make_acado_solver('../acado_nmpc_step', 'model.c')
95 cd ..
```

4.2.1 ROS Integration

As described in Section 3.2.3, we integrate the ACADO solver into a ROS node for generating control solutions in real-time on the aircraft. However, our approach for the fixed-wing UAV differs slightly from the MAV as all low-level control and state estimation is performed on the low-level autopilot's micro-controller (we use the Pixhawk Autopilot [1]). As processing power on the Pixhawk micro-controller is somewhat limited, an additional onboard ODROID-U3 computer with 1.7 GHz Quad-Core processor and 2 GB RAM, running Robotic Operating System (ROS) [3] is integrated into the platform for experimentation with more computationally taxing algorithms. High-level controllers can be run within ROS node wrappers which communicate with the Pixhawk via UART serial communication; average communication latency was observed $<3\,\mu$s.The NMPC is run within a ROS node on the ODROID-U3. MAVROS [21], an extendable communication node for ROS, is used to translate MAVLink Protocol messages containing current state estimates from the Pixhawk, and similarly send back control references from the NMPC implemented in ROS. As high-level fixed-wing dynamics are somewhat slow, we choose a fixed rate loop for control generation using a simple while loop, shown as an example in the following code excerpt.

```
1 while (ros::ok()) {
2
3     /* empty callback queues */
```

```
    ros::spinOnce();

    /* nmpc iteration step */
    ret = nmpc.nmpcIteration();

    if (ret != 0) {
      ROS_ERROR("nmpc_iteration: error in qpOASES QP solver.");
      return 1;
    }

    /* sleep */
    nmpc_rate.sleep();
  }
```

Note the NMPC iteration step is only called once per loop, and the sleep function regulates the timing of the loop. The `ros::spinOnce()` updates any subscriptions with the most recent state estimates for use in the controller, and control action is subsequently published as a `geometry_msgs::TwistStamped` message for processing within the MAVROS attitude setpoint plugin. This plugin is integrated for off-board control functionality within the Pixhawk standard firmware. Messages from the pixhawk are streamed at a rate of ∼40 Hz, and it is reasonable to assume the callback functions will contain up-to-date values in their queues at every NMPC iteration; as mentioned previously, typical high-level control rates for fixed-wing vehicles are often 10 Hz or 20 Hz. All augmented states, i.e. not directly measured or estimated, used within the controller are also stored and kept for the next iteration.

4.2.2 Experimental Results

System identification, controller design, and flight experiments described in this section are peformed on a small, 2.6 m wingspan, light-weight 2.65 kg, low-altitude, and hand-launchable fixed-wing UAV, Techpod, see Figure 19.

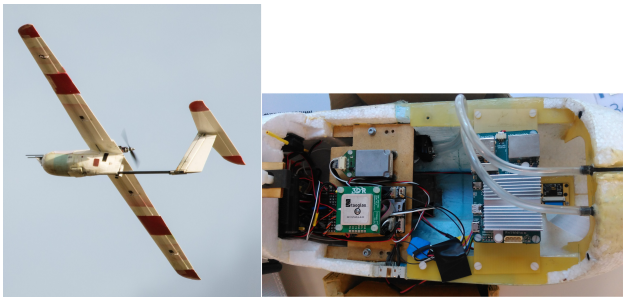


Fig. 19 Fixed-wing Test Platform

Two flight experiments were conducted using the described framework. A horizon length of $N = 40$, or 4 s was used with objective weights $Q_{y_{diag}} = P_{diag} =$

$[0.01, 10, 0.1, 0.01, 10]$, $Q_u = 100$. The discretization time step within the horizon is T_{step}=0.1 s, and the NMPC is iterated every 0.05 s. The average solve time for the NMPC running on the ODROID-U3 was observed to be $\sim$13 ms. Both experiments took place during very calm conditions, and the wind speed was negligible.

In Fig. 4.2.2, Techpod is commanded towards a box pattern until returning to a loiter circle. Minimal overshoot is observed, considering the set acceptance radius of 35 m, and convergence within less than 1 m of position error is observed for each line segment and the final loiter circle. The commanded and actual roll angles as well as the roll rate, are both kept within acceptable bounds.

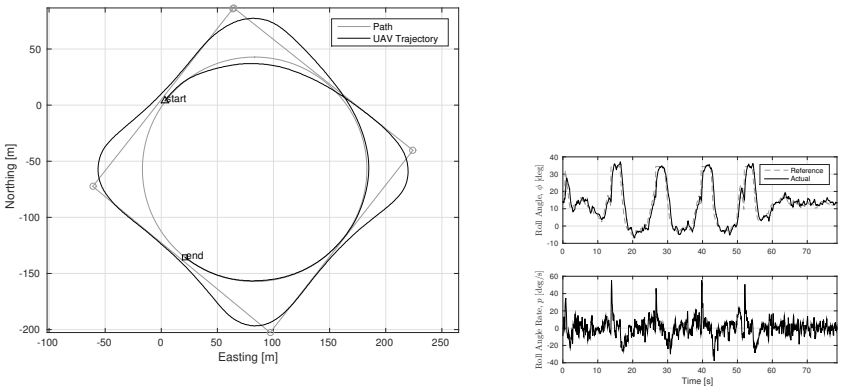


Fig. 20 Flight experiment: box tracking

In Fig. 4.2.2, an arbitrary sequence of Dubins segments were given to the high-level NMPC. Again, good convergence to the path is seen, with acceptable state responses. Position error of less than 3 m was observed after convergence to the path. The end of the shown flight path is stopped just before converging to the final loiter due to rain fall starting during the flight experiment and manual take-over of the aircraft for landing.

5 Conclusion

In this chapter we presented an overview of many model-based control strategies for multiple classes of unmanned aerial vehicles. The strategies presented are: Robust MPC for multi-rotor system, Linear MPC for multi-rotor system and Nonlinear MPC for multi-rotor system and fixed-wing UAVs. These control strategies have been evaluated in real experiments and performance evaluation has been shown in this chapter. The presented controllers are available as open source ROS package on `https://github.com/ethz-asl/mav_control_rw` for rotary wing

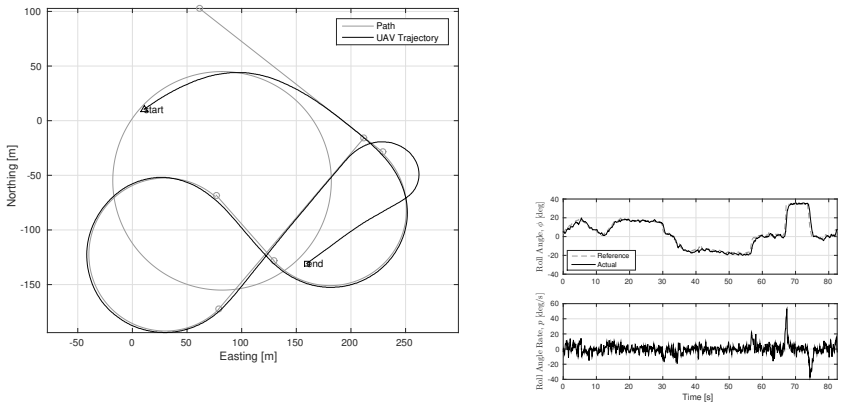


Fig. 21 Flight experiment: Dubins tracking

MAVs and `https://github.com/ethz-asl/mav_control_fw` for fixed wing MAVs.

References

1. "Px4 autopilot." [Online]. Available: https://pixhawk.org
2. "Navio autopilot." [Online]. Available: https://emlid.com
3. "Robot operating system." [Online]. Available: http://www.ros.org
4. D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.
5. S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
6. F. Borrelli, A. Bemporard, and M. Morari, *Predictive Control for Linear and Hybrid Systems*, 2015.
7. H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.
8. K. Alexis, C. Papachristos, R. Siegwart, and A. Tzes, "Robust model predictive flight control of unmanned rotorcrafts," *Journal of Intelligent & Robotic Systems*, pp. 1–27, 2015.
9. J. Loefberg , "Minimax approaches to robust model predictive control," Ph.D. dissertation, Linkoping University, Linkoping, Sweden, 2003.
10. M. Cannon, S. Li, Q. Cheng and B. Kouvaritakis, "Efficient robust output feedback mpc," in *Proceedings of the 18th IFAC World Congress*, vol. 18, 2011, pp. 7957–7962.
11. M. Kvasnica, *Real-Time Model Predictive Control via Multi-Parametric Programming: Theory and Tools*. VDM Verlag, 2009.
12. J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.
13. L. Ljung, *System identification - Theory for the User*. Prentice-Hall, 1999.
14. S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3923–3929.

15. "vicon systems." [Online]. Available: http://www.vicon.com
16. B. Houska, H. Ferreau, M. Vukov, and R. Quirynen, "ACADO Toolkit User's Manual," http://www.acadotoolkit.org, 2009–2013.
17. E. A. Morelli, "Low-order equivalent system identification for the tu-144ll supersonic transport aircraft," *Journal of guidance, control, and dynamics*, vol. 26, no. 2, pp. 354–362, 2003.
18. Y. Luo, H. Chao, L. Di, and Y. Chen, "Lateral directional fractional order (pi) π control of a small fixed-wing unmanned aerial vehicles: controller designs and flight tests," *Control Theory & Applications, IET*, vol. 5, no. 18, pp. 2156–2167, 2011.
19. R. W. Beard and T. W. McLain, "Implementing dubins airplane paths on fixed-wing uavs," *Contributed Chapter to the Springer Handbook for Unmanned Aerial Vehicles*, 2013.
20. T. Stastny, A. Dash, and R. Siegwart, "Nonlinear mpc for fixed-wing uav trajectory tracking: Implementation and flight experiments," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2017.
21. MAVROS, 2016, http://wiki.ros.org/mavros.