Project in Adaptive Control and Real Time Systems

March 7, 2016

1 Introduction

2 Dynamics

In this project, we consider the non-linear quadcopter equations as derived by Lukkonen et al. [1]. A brief description of the dynamics is given to define terms which will be used in the control scheme. Let

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \quad \boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}, \tag{1}$$

where $\boldsymbol{\xi}$ [m] denotes the position of the centre of mass in the global coordinate system, $\boldsymbol{\eta}$ [rad] is the euler-angles in the body coordinate system and ω_i [rad/s] is the angular speed of the rotor i. For future reference, the basis vectors in the cartesian coordinate system are written $\hat{\mathbf{x}}$, and the subindexing $\cdot_B$ refers to the vector of matrix defined in the body coordinate system. The translation from the global- to the body coordinate system is done by the orthogonal rotation matrix

$$\mathbf{R} = \begin{bmatrix} \cos(\psi)\cos(\phi) & \cos(\psi)\sin(\theta)\sin(\phi) - \sin(\psi)\cos(\phi) & \cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) \\ \sin(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\sin(\phi) + \cos(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) - \cos(\psi)\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \tag{2}$$

such that a vector defined in the body system $\mathbf{v}_B$ can be translated to the global coordinate by the mapping

$$\mathbf{v} = \mathbf{R}^{-1}\mathbf{v}_B = \mathbf{R}^T\mathbf{v}_B. \tag{3}$$

The force generated by a rotor is assumed to be proportional to the rotor speed squared,

$$f_i = k\omega_i^2 \tag{4}$$

with some constant k and the torque around each motor axis can be written

$$\tau_{M_i} = b\omega_i^2 + I_M\dot{\omega}_i \tag{5}$$

where b is a drag constant and I_M is the rotor inertia. By the symmetry of the system, the thrust and torque vectors in the body coordinate system can then be wtitten

$$\mathbf{T}_B = T_B \hat{\mathbf{z}}_B = \begin{bmatrix} 0 \\ 0 \\ k \sum_{i=1}^{4} \omega_i^2 \end{bmatrix}, \qquad \boldsymbol{\tau}_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} kl(-\omega_2^2 + \omega_4^2) \\ kl(-\omega_1^2 + \omega_3^2) \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix} \tag{6}$$

Assuming that the quadcopter experiences air resistance, which increases with $\dot{\boldsymbol{\xi}}$, we simply define a matrix

$$\mathbf{D} = \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{bmatrix} \tag{7}$$

where the coefficients remain to be estimated. With the above definitions, the non-linear dynamics of the quadcopter can then be derived from the Newton-Euler equations as

$$\begin{cases} m\ddot{\boldsymbol{\xi}} = m\mathbf{G} + \mathbf{T}_B - \mathbf{D}\dot{\boldsymbol{\xi}} \\ \ddot{\boldsymbol{\eta}} = \mathbf{J}^{-1}(\boldsymbol{\eta})(\boldsymbol{\tau}_B - \mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}}), \end{cases} \tag{8}$$

A brief description of **J** and **C** matrices can be found in **Section 6**, but the interested reader is referred to [1] for a more thorough derivation of the Newton-Lagrange equations. In the work of Lukkonen, this system was simulated in continuous time, and here we will take an alternate approach in order to implement the dynamics as a discrete time ROS node in Python.

2.1 Non-linear model

By defining the states and control signals as

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\xi} \\ \dot{\boldsymbol{\xi}} \\ \boldsymbol{\eta} \\ \dot{\boldsymbol{\eta}} \end{bmatrix} \in \mathbb{R}^{12\times 1}, \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \in \mathbb{R}^{4\times 1} \tag{9}$$

respectively, the full non-linear system can then be written

$$\begin{aligned} \dot{\mathbf{x}}(t) =& \mathbf{A}^c\mathbf{x}(t) + \mathbf{B}^c\mathbf{u}(t) + \mathbf{G}^c \\ \mathbf{y}(t) =& \mathbf{C}^c\mathbf{x}(t) \end{aligned} \tag{10}$$

with

$$\mathbf{A}^c = \begin{bmatrix} \mathbf{0} & \mathbb{I}_{3\times 3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\frac{1}{m}\mathbf{D} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbb{I}_{3\times 3} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{J}^{-1}(\boldsymbol{\eta})\mathbf{C}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) \end{bmatrix}, \mathbf{B}^c = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \frac{1}{m}\mathbf{R}\hat{\mathbf{z}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}(\boldsymbol{\eta})^{-1} \end{bmatrix}, \mathbf{G}^c \begin{bmatrix} \mathbf{0} \\ \mathbf{G} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \mathbf{C}^c = \begin{bmatrix} \mathbf{0}_{7\times 2} & \mathbb{I}_{7\times 7} & \mathbf{0}_{7\times 2} \end{bmatrix} \tag{11}$$

The $\mathbf{C}^c$ matrix was chosen to reflect the available sensory information. The height z is measured by a pressure sensor, the angles $\boldsymbol{\eta}$ are retrieved from a gyroscope aboard the quadcopter and the velocities $\dot{\boldsymbol{\xi}}$ are integrated from readings of the accelerometer.

In order to simulate the dynamics, the continuous time system (10) was implemented in Simulink (see `quadcopter_model.m`, **Section 6**), and validated by comparison to the results in [1]. The discrete time system was then computed using zero-order hold at a time step h, with the discrete state space representation

$$x(t_k + h) = \mathbf{A}^d x(t_k) + \mathbf{B}^d \mathbf{u}(t_k) + \mathbf{G}^d \tag{12}$$

$$y(t_k) = \mathbf{C}^d x(t_k) \tag{13}$$

$$\tag{14}$$

where

$$\mathbf{A}^d = e^{\mathbf{A}^c h}, \qquad \mathbf{B}^d = \int_0^h e^{\mathbf{A}^c h} ds B^c, \qquad \mathbf{G}^d = h\mathbf{G}^c, \qquad \mathbf{C}^d = \mathbf{C}^c. \tag{15}$$

As the final implementation of the control system was done in ROS based in Python and C++, a script was written do simulate the system using only the scipy and numpy modules (see simulate_system.py, **Section 6**). The result (see Figure **??**).

2.2 Full model with baseline PD dynamics

3 MPC control

The dynamics of the quadcopter defined in [2] defines the pitch as θ_1 and yaw as θ_2. The quadcopter is then linearised round $(\theta_1, \theta_2) = \mathbf{0}$ with the control signals $u_1 = \Delta\theta_1$, $u_2 = \Delta\theta_2$ and u_3 as the commanded thrust (see `quadcopter_MPC_init.m`). Discretisation is here done through ZOH but other methods could be used as well. The model was replicated in Simulink, and the system responses look reasonable (see `quadcopter_MPC_simulate.m`)

TODO

1. ~~Create simplified linearised system ss-model for use in MPC~~ (see eg. [2]).
2. ~~Validate by comparison to the results in~~ [2].
3. ~~Set up QP-MPC controller with Simulink MPC-library~~ (see eg. [2]).
4. ~~Validate by comparison to the results in~~ [2].
5. Set up QP-MPC controller with CVXGEN m-code (see eg. [2] [3]).
6. Validate by comparison to the results in Simulink.
7. System identification.
8. Simulate system with proper parameters.

4 $\mathcal{L}_1$-control

The $\mathbf{\Gamma}$-projection operator for two vectors $\theta, y \in \mathbb{R}^k$ is defined as

$$\mathrm{Proj}_{\mathbf{\Gamma}}(\theta, y, f) = \begin{cases} \mathbf{\Gamma}y - \mathbf{\Gamma}\dfrac{\nabla f(\theta)(\nabla f(\theta))^T}{||\nabla f(\theta)||_2}\mathbf{\Gamma}y f(\theta) & \text{if } \ f(\theta) > 0 \ \text{ and } \ y^T\nabla f(\theta) > 0 \\ \mathbf{\Gamma}y & \text{otherwise.} \end{cases} \tag{16}$$

where $\mathbf{\Gamma} = \mathbb{I}_{k\times k}\Gamma$ for some scalar $\Gamma > 0$ (typically $\Gamma \approx 10^5$) and $f(\theta)$ is a convex function [4]. By solving the Lyapunov equation

$$\mathbf{A}_m\mathbf{X} + \mathbf{X}\mathbf{A}_m^T + \mathbf{Q} = 0, \tag{17}$$

for $\mathbf{P} = \mathbf{P}^T$, with some arbitrary $\mathbf{Q} > 0$, the feedback controller

$$\begin{cases} u(t) = \hat{\theta}^T x(t) + k_g r(t) \\ \dot{\hat{\theta}}(t) = \mathrm{Proj}_{\mathbf{\Gamma}}(\hat{\theta}^T(t), x(t)\tilde{x}^T(t)\mathbf{X}b) \end{cases} \tag{18}$$

can be constructed, where $\tilde{x} = \hat{x} - x$ is the state estimation error, k_g is a gain and $r(t)$ is the reference signal. By designing the companion system

$$\begin{cases} \dot{x}(t) = \mathbf{A}_m\hat{x}(t) + b(u(t) - \hat{\theta}^T(t)x(t)) \\ y(t) = c^T\hat{x}(t) \end{cases} \tag{19}$$

it can be shown (by Theorem 2 [5]) that the state estimation error,

$$\lim_{t\to\infty} \tilde{x} = 0. \tag{20}$$

By a corollary of the theorem, choosing

$$k_g = -\frac{1}{c^T\mathbf{A}_m^{-1}b} \Rightarrow \lim_{t\to\infty} y(t) = r \tag{21}$$

if $r \equiv$ constant.

TODO

1. ~~Define general control structure.~~
2. Create Simulink projection operator (see eg. [6])
3. Validate projection operator against benchmark Simulink models (eg. [5]).
4. Define robustness metrics (see eg. [6] [7])
5. Create script for computing the $\mathcal{L}_1$-gain (see eg. [6]).
6. Validate script against benchmark Simulink models (eg. [5]).
7. Simulate control.

5 General TODOs

1. Found that one of the copters were broken - sent to Bitcraze for repairs, expected to be done in early march.

2. Tried installing IRIS in Ubuntu but ran into issues using the PODS ”make” command required to get everything up and running. TODO: contact Claes or Anders to get help in finding someone experienced with PODS.

References

[1] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, 2011.

[2] P. Bouffard, "On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments," Master's thesis, EECS Department, University of California, Berkeley, Dec 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-241.html

[3] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[4] E. Lavretsky, T. E. Gibson, and A. M. Annaswamy, "Projection operator in adaptive systems," 2011.

[5] C. Cao and N. Hovakimyan, "Design and analysis of a novel l1 adaptive controller, part i: Control signal and asymptotic stability," in *American Control Conference, 2006*. IEEE, 2006, pp. 3397–3402.

[6] N. Hovakimyan, "L1 tutorial." [Online]. Available: http://naira-hovakimyan.mechse.illinois.edu/l1-adaptive-control-tutorials/

[7] M. Q. Huynh, W. Zhao, and L. Xie, "L 1 adaptive control for quadcopter: Design and implementation," in *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*. IEEE, 2014, pp. 1496–1501.

6 Appendix