The Desktop Quad

with an LQR setpoint controller

Parker Lusk

April 21, 2017

Abstract

This report describes the progress of the Desktop Quad achieving fully autonomous flight.

1 Introduction

The Desktop Quad (see Figure 1) is an effort to allow an inexpensive quadrotor to be used as a case study in controls, vision processing, and autopilot design scenarios. It features an 8cm x 8cm quadrotor with an upward-facing camera for localization. The (really) micro air vehicle (MAV) is tethered using flexible silicone wire for communications and power, allowing indefinite flight. This project aims to close the attitude loop by adding a linear-quadratic regulator (LQR) setpoint controller.

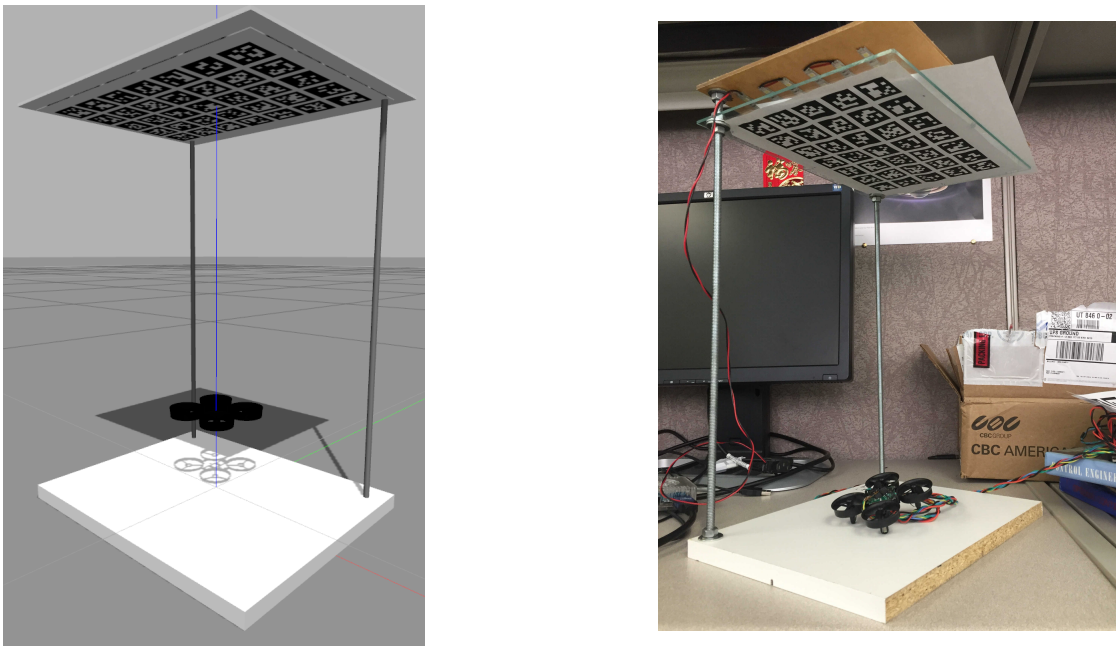


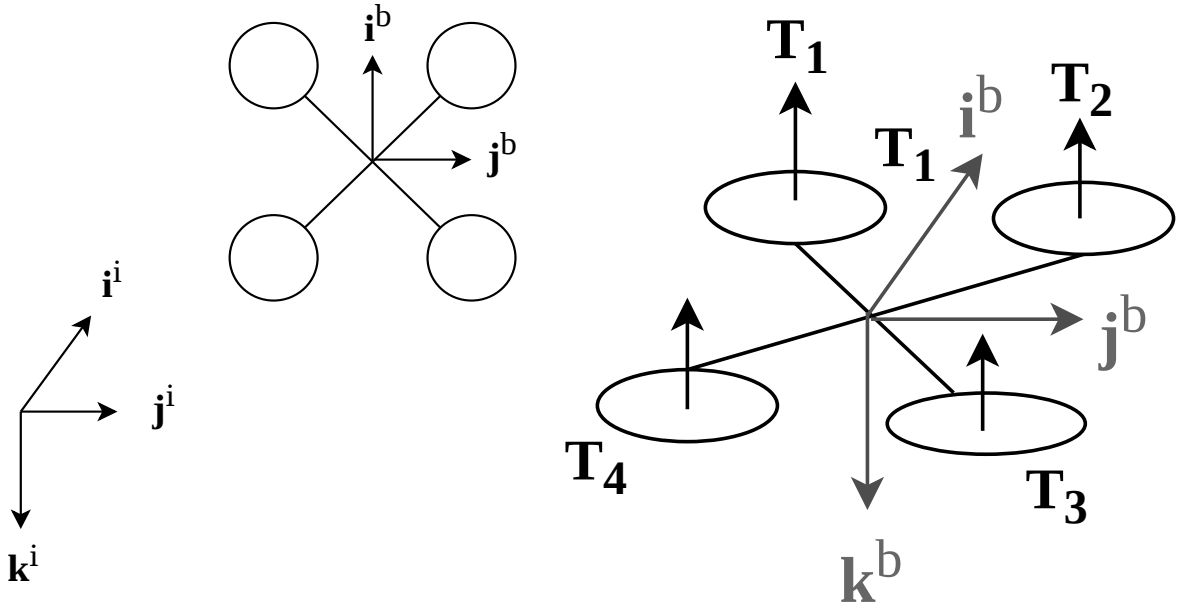
Figure 1: The Desktop Quad simulated in Gazebo/ROS, next to the hardware platform.

2 System Modeling

We begin by modeling the Desktop Quad multirotor and deriving the equations of motion. The 12 state variables that will be used to describe the equations of motions are

$$x = \begin{bmatrix} p_n & p_e & p_d & u & v & w & \phi & \theta & \psi & p & q & r \end{bmatrix}^T$$

where p_n, p_e, and p_d are inertial north, east, down coordinates; u, v, w are body frame velocities; ϕ, θ, ψ are Euler angles; and p, q, r are body frame angular rates. The two relevant coordinate frames for system modeling are shown in Figure 2a.



(a) Relevant coordinate frames for modeling

(b) Forces modeled on the quadrotor

Figure 2: Quadrotor modeling diagrams.

2.1 Kinematics and Dynamics

In order to analyze, simulate, and control the Desktop Quad, a 6DOF dynamic model must be derived. This is done using kinematics and dynamics. The goal is to find the evolution equations for each of the quantities we care about, as found in the state vector above [1].

$$\begin{bmatrix} \ddot{p}_n \\ \ddot{p}_e \\ \ddot{p}_d \end{bmatrix} = R_b^i(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \frac{1}{m} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s_\psi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & \frac{s_\phi}{c_\theta} & \frac{c_\phi}{c_\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \Gamma_1 pq - \Gamma_2 qr \\ \Gamma_5 pr - \Gamma_6 (p^2 - r^2) \\ \Gamma_7 pq - \Gamma_1 qr \end{bmatrix} + \begin{bmatrix} \Gamma_3 l + \Gamma_4 n \\ \frac{1}{J_y} m \\ \Gamma_4 l + \Gamma_8 n \end{bmatrix} \tag{3}$$

where $T = T_1 + T_2 + T_3 + T_4$, m is the mass of the multirotor, and the trigonometric functions are shortened as shown.

3 Control Architecture

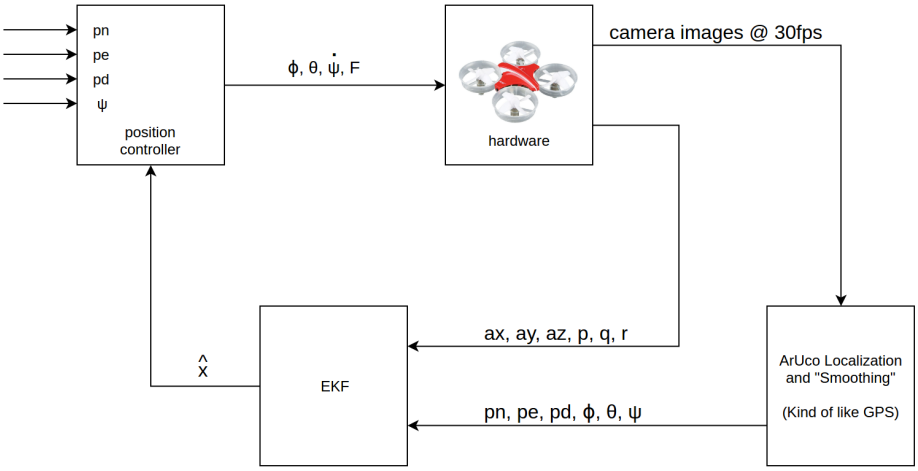


Figure 3: Control architecture of the Desktop Quad system.

The Desktop Quad uses the ROSflight [4] stack for autonomous flight and thus has an on-board autopilot for attitude control. This simplifies the control problem and eliminates the need to control the moments l, m, and n. The LQR controller developed here will close the attitude loop using a position setpoint controller as shown in Figure 3. The simplified outer-loop state vector is defined as

$$x = \begin{bmatrix} p_n & p_e & p_d & \dot{p}_n & \dot{p}_e & \dot{p}_d & \psi \end{bmatrix}^T$$

with input to the ROSflight autopilot

$$\nu = \begin{bmatrix} T & \phi & \theta & r \end{bmatrix}^T.$$

Because of how we have defined our state, we will need to create a nonlinear map between the control u produced by our controller and the input ν to the autopilot. We can use the kinematic and dynamic relationships to define the map

$$\boldsymbol{u} = \begin{bmatrix} u_{p_{3x1}} \\ u_{\psi_{1x1}} \end{bmatrix}, \tag{4}$$

where

$$u_p = R_b^i(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \frac{1}{m} \tag{5}$$

and

$$u_\psi = q\frac{sin\phi}{cos\theta} + r\frac{cos\phi}{cos\theta}. \tag{6}$$

Inverting these relationships will give us a mapping for how u affects ν, which can be feed into autopilot. The state-space system can be represented as

$$\dot{x} = Ax + Bu + bg \tag{7}$$

with

$$A = \begin{bmatrix} 0_{3x3} & I_{3x3} & 0_{3x1} \\ 0_{3x3} & 0_{3x3} & 0_{3x1} \\ 0_{1x3} & 0_{1x3} & 0_{1x1} \end{bmatrix}, \quad B = \begin{bmatrix} 0_{3x3} & 0_{3x1} \\ I_{3x3} & 0_{3x1} \\ 0_{1x3} & 1 \end{bmatrix}, \quad b = [0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0]^T. \tag{8}$$

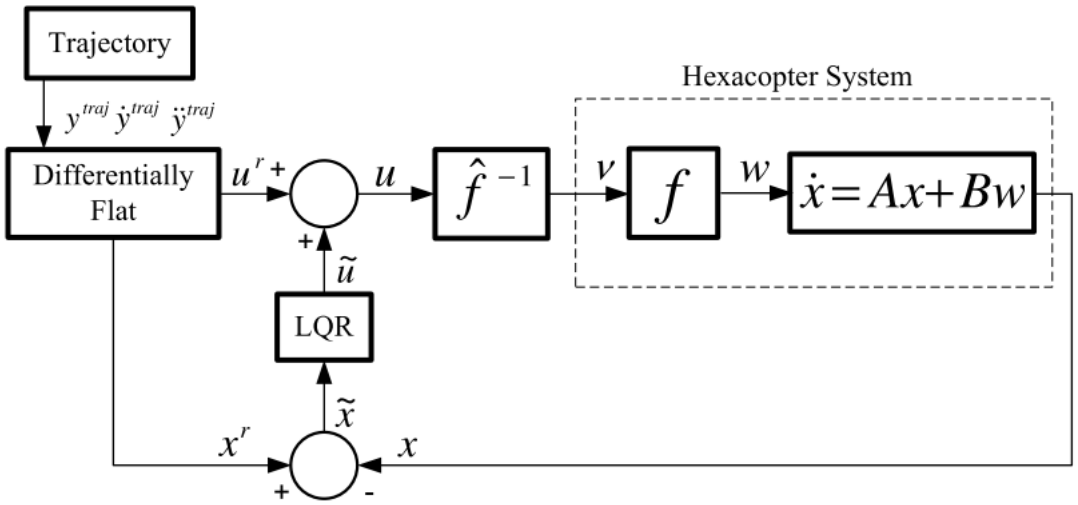


Figure 4: Differentially flat / LQR controller, shamefully stolen from [3].

4 Differential Flatness

A differentially flat system is one in which the state and control inputs can be expressed as functions of the output and its time derivatives. Formulating the problem this way let's us use these properties to create a 2nd order trajectory that the multirotor will follow. If our models were perfect, the differentialy flat block of Figure 4 would only need to output a control u that then would go into the model. Because we should never take our models (or ourselves) too seriously, we add LQR to regulate our error state to 0.

5 Simulation Environment

The LQR setpoint controller was first designed in MATLAB / Simulink based on the MAGICC lab Mikrocopter Hexacopter simulation. Along with the setpoint controller, a differential flatness [3] trajectory generator was implemented to compare and contrast against. Figure 5 shows the Simulink diagram and how the differential flatness trajectory or the setpoint can be selected during the simulation.

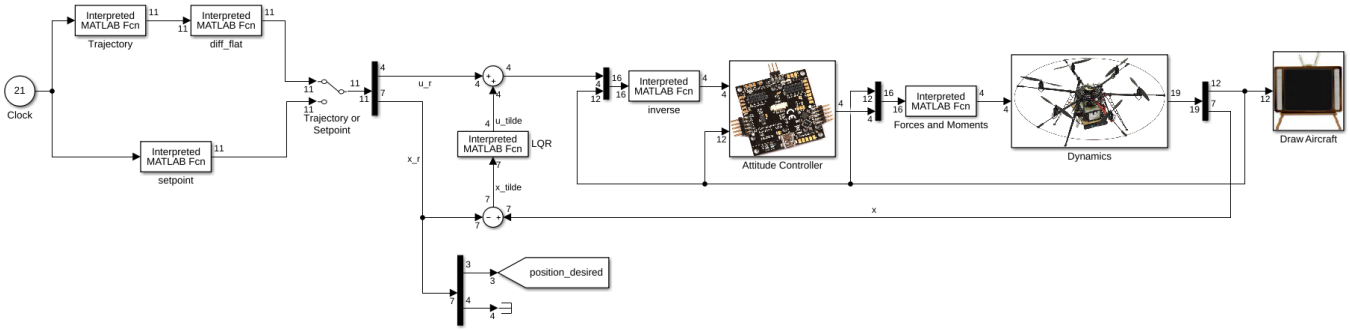


Figure 5: The LQR setpoint controller design simulated in MATLAB/Simulink.

After developing and tuning the control algorithm in MATLAB using the `lqr` command, the design and gains were then ported to C++ for ROS/Gazebo. Using ROS allows the same implementation to be used in the Desktop Quad simulator and directly in hardware (see Figure 1).

6 Results

Results of using the setpoint and differential flatness controllers can be found in Figures 6 and 7 respectively. Video results of the MATLAB and ROS simulations can be shown here and here.

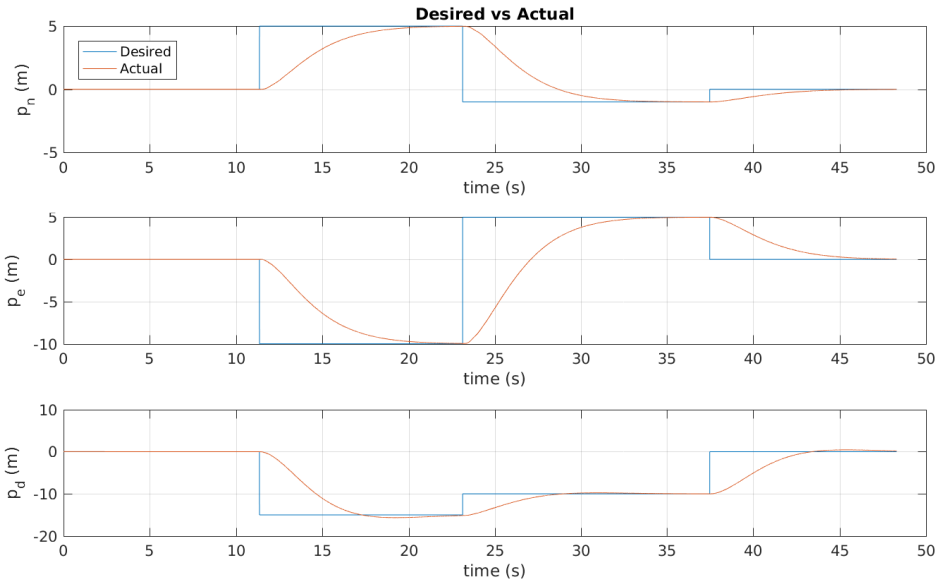


Figure 6: Error of the commanded and actual positions using the setpoint controller.

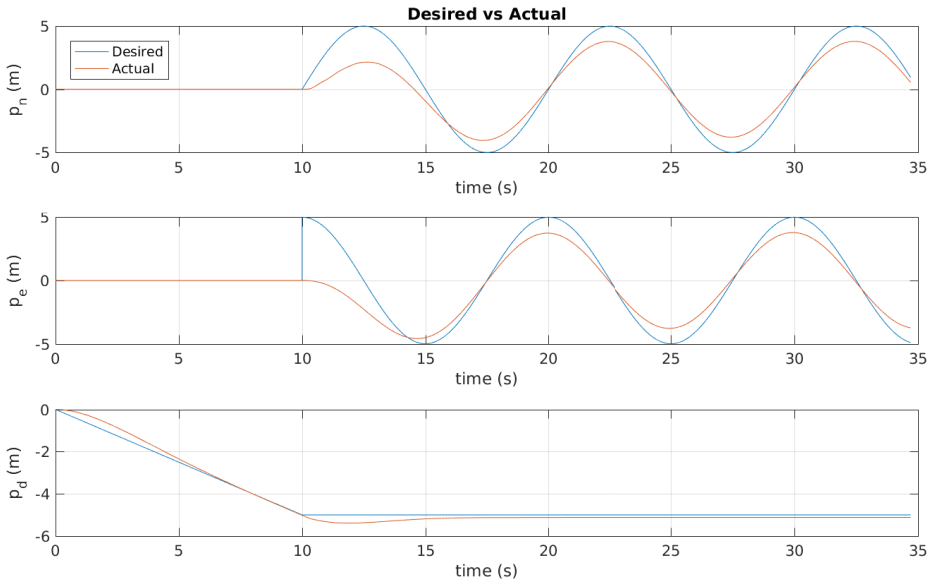


Figure 7: Error of the commanded and actual positions using the differential flatness trajectory generator.

7 Conclusion

Using the differential flatness properties of the quadrotor system allows intuitive control of the outer position loop. As a result of using LQR to regulate the error state $\tilde{x}$ to 0, the controlled motion is very smooth and stable.

It was good for me to implement the differentially flat LQR controller in MATLAB and C++. I was able to further understand the importance and use of the feedforward gain N and the state gain F.

I noticed in the Gazebo simulation that I had more steady-state error than in the MATLAB simulation. I would like to add an integrator to the position states to further improve my control accuracy.

References

[1] R. W. Beard and T. W. McLain, "Small unmanned aircraft," 2011.

[2] R. W. Beard and T. W. Mclain, "Introduction to Feedback Control using Design Studies," 2016.

[3] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," IEEE Int. Conf. Intell. Robot. Syst., pp. 2688–2693, 2011.

[4] J. Jackson, G. Ellingson, and T. McLain, "ROSflight: A lightweight, inexpensive MAV research and development tool," 2016 Int. Conf. Unmanned Aircr. Syst. ICUAS 2016, pp. 758–762, 2016.