2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Onboard State Dependent LQR for Agile Quadrotors

Philipp Foehn, Davide Scaramuzza

Abstract— **State-of-the-art approaches in quadrotor control split the problem into multiple cascaded subproblems, exploiting the different time scales of the rotational and translational dynamics. They calculate a desired acceleration as input for a cascaded attitude controller but omit the attitude dynamics. These approaches use limits on the desired acceleration to maintain feasibility and robustness through the control cascade. We propose an implementation of an LQR controller, which: (I) is linearized depending on the quadrotor's state; (II) unifies the control of rotational and translational states; (III) handles time-varying system dynamics and control parameters. Our implementation is efficient enough to compute the full linearization and solution of the LQR at a minimum of $10\,\mathrm{Hz}$ on the vehicle using a common ARM processor. We show four successful experiments: (I) controlling at hover state with large disturbances; (II) tracking along a trajectory; (III) tracking along an infeasible trajectory; (IV) tracking along a trajectory with disturbances. All the experiments were done using only onboard visual inertial state estimation and LQR computation. To the best of our knowledge, this is the first implementation and evaluation of a state-dependent LQR capable of onboard computation while providing this amount of versatility and performance.**

Supplementary Material

This paper is accompanied by a video showcasing the conducted experiments: `https://youtu.be/8OVsJNgNfa0`

I. Introduction

A. Motivation

With the recent advances in control of Micro Aerial Vehicles (MAVs), it is possible to use them in a wide variety of applications, ranging from search and rescue scenarios, observation, hobbyist drone racing to transportation and delivery. Many of these scenarios include difficult, cluttered environments, such as post-disasters or emergency situations. In all these applications, robust control of the vehicle plays a crucial role for success, which becomes particularly challenging considering the quadrotor as an under-actuated system with sublime agility and simplicity.

Many groups have shown examples of complex, agile maneuvers [1], [2], [3], [4], [5] which rely on excellent tracking of a given trajectory. This tracking control is often based on assumptions and simplifications, exploiting the time-scale separation between the translational and rotational dynamics, but introducing certain limitations.

The authors are with the Robotics and Perception Group, Dep. of Informatics, University of Zurich, and Dep. of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland—`http://rpg.ifi.uzh.ch`. This research was supported by the National Centre of Competence in Research (NCCR) Robotics, through the Swiss National Science Foundation, the SNSF-ERC Starting Grant, and the DARPA FLA program.

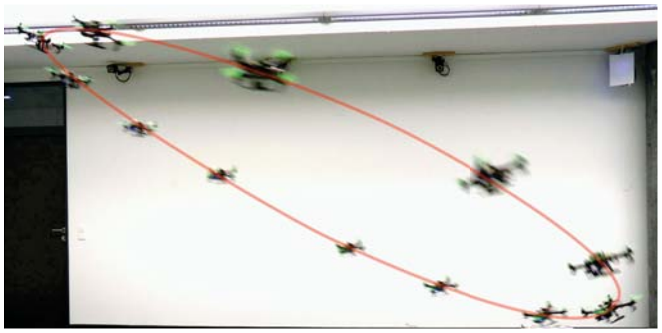
Fig. 1. Flying an ellipse with sate-dependent LQR control. Our quadrotor is based on a Qualcomm Snapdragon Flight board with the Pixhawk PX4 flight stack, onboard visual inertial odometry and commercial electronic speed controllers, motors, propeller and frame. It has a take-off weight of $255\,\mathrm{g}$.

Richard Bellmans work on dynamic programming [6], [7] and Rudolf Kalmans optimal control theory [8] formed the realm of the Linear Quadratic Regulator (LQR), a powerful tool towards optimal control. Until today, LQR has often only been used to solve parts of the control problem on MAVs, such as attitude stabilization, but never to provide a full-state controller unifying rotational and translational states in one feedback loop. Exactly this unified control scheme is where LQR can exceed other approaches.

B. Related Work & Problem Statement

State-of-the-art approaches in control and trajectory generation exploit the time-scale separation and differential flatness of these systems [9]. They rely heavily on cascaded control schemes [10] and *decouple* the rotational and translational dynamics via a geometric tracking controller as described in [11]. These control schemes leverage the desired acceleration to prescribe the attitude, which therefore is directly affected by the over-imposed position and velocity control. Direct control over the attitude itself is therefore lost and the demanded acceleration is subject to several limitations in change rate, magnitude, and direction.

On the other hand, many approaches on LQR [12] linearize the system at a given stable state [13] or use a precomputed library of LQR gains [14]. None of these approaches is capable of adjusting to the full state space, varying state or input costs, or changing system parameters at execution time. Moreover, they often separate orientation and position control, similarly to the aforementioned geometric control. Also for Model Predictive Control (MPC), this decoupling is used to simplify the problem [15], [16].

A reason for this is surely the difficulty of applying an LQR scheme to a system where inputs (body-frame) and states (world-frame) do not share the same reference coordinate frame. In the example of most MAVs, this means that the attitude radically changes the linearization, even in the simplified case of a quadrotor where often only the yaw state is concerned. This necessitates the recomputation of the LQR whenever an attitude change occurs, or in simplified cases at a certain rate, to handle this state-dependent linearization.

All of these methods make use of the under-actuation property of the system, splitting the control problem into multiple subproblems. Especially leveraging the desired acceleration to define the attitude of the vehicle leads to complications, since it disregards the attitude dynamics and therefore requires planned smooth trajectories which must also account for actuator saturations. One specific problem arising from this control scheme is the difficulty to distinguish between actually desired negative downward acceleration (wanted), and erroneously large downward acceleration from large position errors (unwanted). This makes it difficult to use one single control algorithm for many different maneuvers, while keeping the interface and tuning possibilities simple and intuitive for the user. Neither could varying system-and-control parameters be handled without significant complications.

C. Contribution

We propose a linear quadratic regulator that unifies the control of translational and rotational dynamics of a quadrotor and mitigates many of the aforementioned shortcomings. It vastly simplifies the tuning of the system, since the tuning parameters are costs that directly relate to state errors, weighting *all* system states in relation to each other. To achieve this, we leverage the formulation of dynamics, reference states, trajectories, and costs to implement a state-dependent LQR.

This provides a robust way to control such a MAV not only in the reduced static state space (hover) but also along dynamic state sequences (trajectories). The shown formulation of the dynamics can be extended with a more detailed state description, for example the dynamics of inputs or even virtual states, like the integral of position error. We focus here on the discrete infinite time horizon LQR solution, but our linearization method translates without restrictions to iterative LQR or other model predictive control schemes.

In multiple experiments, we show the robustness of our controller in various situations, such as disturbance from hover, reference jumps, tracking of feasible as well as infeasible trajectories and disturbance during such tracking. We use a small MAV as visible in Fig. 1 with all computation running onboard.

D. Structure of the Paper

We first give an overview of the control architecture, in Sec. II-A, since this defines our system dynamics formulation, which is described in the following Sec. II-B. The LQR control principle is outlined in Sec. II-C with a detailed explanation of the linearization of this system in Sec. II-D. We explain the tracking scheme used for the experiment

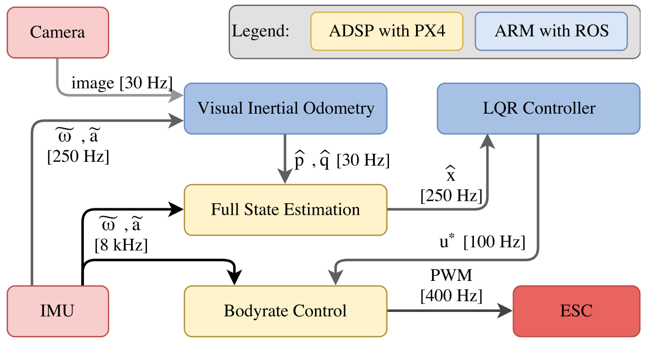

Fig. 2. Overeview of the control architecture implementation on the Snapdragon Flight with measured acceleration $\tilde{\boldsymbol{a}}$ and bodyrates $\tilde{\boldsymbol{\omega}}$, estimated position $\hat{\boldsymbol{p}}$, orientation $\hat{\boldsymbol{q}}$ and full state $\hat{\boldsymbol{x}}$, and desired inputs $\boldsymbol{u}^*$

in Sec. II-E. The experiment setup, platform and control architecture is specified in Sec. III. We show our results in Sec. IV and comment our findings the discussion Sec. V followed by the conclusion Sec. VI.

II. Methodology

A. Control Hierarchy

Most quadrotors are equipped with an Inertial Measurement Unit (IMU) to get high-frequency intrinsic measurements of the rotational velocity and linear acceleration to stabilize the rotational dynamics, and additional extrinsic information to stabilize the translation dynamics. This extrinsic information often comes from GPS, offboard motion capture systems or nowadays also from onboard visual (-inertial) odometry, where we will focus on the last. While the inertial sensors often work at very high frequencies, of up to 8 kHz, to stabilize the fast rotational dynamics, visual odometry operates at a slower rate (typically 30 Hz, visualized in Fig. 2 and explained in Sec. III-B). This is still enough to estimate position and velocity since state-of-the-art visual inertial odometry fuses measurements from the IMU and the camera to output a higher frequency pose estimate.

This choice of sensors is fitted specifically to multirotors. Since they produce thrust forces and drag torques with each rotor, as long as all rotors lie in one plane, one can simplify these forces to a collective thrust and a torque around each body axis. To control acceleration, velocity and position, the quadrotor must adjust its orientation since the collective thrust is always aligned with the body z-axis.

We can now split the control scheme into two domains of fast and slow dynamics. The fast dynamics contains the bodyrates that can be measured directly by the gyroscope in the IMU. The slow dynamics contains the orientation—which is the integral state of the body rates—and the position and velocity—which are integral states of the acceleration—depending on collective thrust, orientation, and gravity. Since the bodyrates are the integral state of the torque produced by different single motor thrusts, they can be controlled with simple feedback loops using the directly available bodyrate measurement. From a control perspective, we intuitively separate these two domains, where the output of the slow dynamics domain are the desired bodyrates and collective thrust. The fast dynamics domain controls the single rotor thrusts to achieve the desired values. This cascade vastly simplifies the control architecture but our approach fully extends to a system where both domains are fused and controlled together. Note that this does not imply decoupling of the rotational and translational dynamics, but using the first derivative of the rotation as an input to the system.

B. System Dynamics

The system dynamics of a quadrotor can be described as a single rigid-body system. Based on the assumption that the low-level bodyrate controller is faster then our LQR controlled state, we use the bodyrates as input to our system. The bodyrates ${}_B\boldsymbol{\omega}$ are defined in the body-fixed frame, as well as the collective normalized thrust ${}_B\boldsymbol{c} = [0, 0, c]$ whereas gravity $\boldsymbol{g} = [0, 0, g]^T$ is defined in world frame. Furthermore the state $\boldsymbol{x}$ consists of position $\boldsymbol{p}$, orientation $\boldsymbol{q}$ and velocity $\boldsymbol{v}$.

$$\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}]^T = [p_x, p_y, p_z, q_w, q_x, q_y, q_z, v_x, v_y, v_z]^T \quad (1)$$

$$\boldsymbol{u} = [{}_B\boldsymbol{\omega}_{des}, {}_B c_{des}]^T = [\omega_{des,x}, \omega_{des,y}, \omega_{des,z}, c_{des}] \quad (2)$$

We use quaternions to represent the orientation of the quadrotor to avoid singularities (gimbal lock) due to angle representation. We will drop the prefix ${}_B[\quad]$ for the body frame from now on. The system dynamics can then be described in a simplified form as:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \qquad \dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \qquad \dot{\boldsymbol{v}} = \boldsymbol{g} + \boldsymbol{q} \odot \boldsymbol{c}, \quad (3)$$

where $\otimes$ is the quaternion multiplication and $\odot$ is the quaternion rotation as:

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \boldsymbol{Q}^\times(\boldsymbol{q}_1)\boldsymbol{q_2} = \bar{\boldsymbol{Q}}^\times(\boldsymbol{q}_2)\boldsymbol{q_1} \quad (4)$$

$$\boldsymbol{q}_1 \odot \boldsymbol{v} = \boldsymbol{R}^\times(\boldsymbol{q})\boldsymbol{v}. \quad (5)$$

$\boldsymbol{Q}^\times(\boldsymbol{q})$ is the quaternion multiplication matrix and $\boldsymbol{R}^\times(\boldsymbol{q})$ is a rotation matrix with

$$\boldsymbol{Q}^\times(\boldsymbol{q}) = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix} \quad (6)$$

$$\bar{\boldsymbol{Q}}^\times(\boldsymbol{q}) = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & q_z & -q_y \\ q_y & -q_z & q_w & q_x \\ q_z & q_y & -q_x & q_w \end{bmatrix} \quad (7)$$

$$\bar{\boldsymbol{Q}}^\times(\boldsymbol{q}) = \boldsymbol{Q}^\times(\bar{\boldsymbol{q}}) \quad (8)$$

$$\bar{\boldsymbol{Q}}^{\times T}(\boldsymbol{q})\boldsymbol{Q}^\times(\boldsymbol{q}) = \begin{bmatrix} 1 & \mathbb{0} \\ \mathbb{0} & \boldsymbol{R}^\times(\boldsymbol{q}) \end{bmatrix}. \quad (9)$$

This results in the full equations for velocity and orientation as:

$$\dot{\boldsymbol{v}} = \boldsymbol{g} + \boldsymbol{R}^\times(\boldsymbol{q})\boldsymbol{c} = \begin{bmatrix} 2(q_w q_y + q_x q_z)c \\ 2(q_y q_z - q_w q_x)c \\ -g + (1 - 2q_x^2 - 2q_y^2)c \end{bmatrix} \quad (10)$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\bar{\boldsymbol{Q}}^\times\left(\begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}\right)\boldsymbol{q} = \frac{1}{2}\begin{bmatrix} -\omega_x q_x - \omega_y q_y - \omega_z q_z \\ \omega_x q_w + \omega_z q_y - \omega_y q_z \\ \omega_y q_w - \omega_z q_x + \omega_x q_z \\ \omega_z q_w + \omega_y q_x - \omega_x q_y \end{bmatrix} \quad . \quad (11)$$

C. Linear Quadratic Regulator

A Linear Quadratic Regulator provides the optimal solution for a given linear time-invariant system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \quad (12)$$

at the reference state $\boldsymbol{x}_0$ and input $\boldsymbol{u}_0$ with respect to a user-defined quadratic cost given by the two positive-definite cost matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. We define the cost-to-go for the infinite-time solution as:

$$\boldsymbol{J}(\boldsymbol{x}, \boldsymbol{u}) = \int_0^\infty \tilde{\boldsymbol{x}}^T \boldsymbol{Q} \tilde{\boldsymbol{x}} + \tilde{\boldsymbol{u}}^T \boldsymbol{R} \tilde{\boldsymbol{u}} \; dt \quad (13)$$

with the errors $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_0$ and $\tilde{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u}_0$. Assume that the optimal cost-to-go is of the quadratic form $\boldsymbol{J}^*(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P} \boldsymbol{x}$. The solution for $\boldsymbol{P}$ can be found using the Continuous Algebraic Riccati Equation

$$0 = \boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} + \boldsymbol{Q} \quad (14)$$

and results in

$$\boldsymbol{u} = \boldsymbol{u}_0 + \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{x}_0) \quad \text{with } \boldsymbol{K} = -\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} \quad (15)$$

as the optimal feedback-policy. To solve the problem, we use a dynamic programming approach as described in [12].

D. Linearization

To calculate the LQR, the full linearization of the system is needed. Instead of just calculating all partial derivatives, we see from (3) that most entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ are independent:

$$\boldsymbol{A}(\boldsymbol{q}, \boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{q}}\boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}) = \begin{bmatrix} \mathbb{0} & \mathbb{0} & \frac{\partial}{\partial \boldsymbol{v}}\dot{\boldsymbol{p}} \\ \mathbb{0} & \frac{\partial}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} & \mathbb{0} \\ \mathbb{0} & \frac{\partial}{\partial \boldsymbol{q}}\dot{\boldsymbol{v}} & \mathbb{0} \end{bmatrix} \quad (16)$$

$$\boldsymbol{B}(\boldsymbol{q}, \boldsymbol{u}) = \frac{\partial}{\partial \boldsymbol{u}}\boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u}) = \begin{bmatrix} \mathbb{0} & 0 \\ \frac{\partial}{\partial \boldsymbol{\omega}}\dot{\boldsymbol{q}} & 0 \\ \mathbb{0} & \frac{\partial}{\partial c}\dot{\boldsymbol{v}} \end{bmatrix}. \quad (17)$$

Therefore, we calculate only the required derivatives in the following sections.

Preface: Partial Derivatives w.r.t. Unit Quaternions: Since the orientation is represented with a unit quaternion , this induces a constraint on the respective states so that $\|\boldsymbol{q}\| = 1$. When deriving a function of this unit quaternion, the constraint does no longer hold for the derivative. To make the constraint generally valid, we discard it for the general quaternion $\boldsymbol{q}$ and enforce a specific unit quaternion $\boldsymbol{q}_u = \boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1}$. The derivation then yields:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{q}} f(\boldsymbol{q}_u) &= \frac{\partial}{\partial \boldsymbol{q}_u} f(\boldsymbol{q}) \cdot \frac{\partial}{\partial \boldsymbol{q}}\left(\boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1}\right) \\ \frac{\partial}{\partial \boldsymbol{q}}\left(\boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1}\right) &= \frac{\partial}{\partial \boldsymbol{q}}\boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-1} + \boldsymbol{q} \cdot \frac{\partial}{\partial \boldsymbol{q}}\|\boldsymbol{q}\|^{-1} \\ &= \|\boldsymbol{q}\|^{-1} \cdot \mathbb{1} - \boldsymbol{q} \cdot \|\boldsymbol{q}\|^{-2} \cdot \frac{\partial}{\partial \boldsymbol{q}}\|\boldsymbol{q}\| \\ &= \|\boldsymbol{q}\|^{-1}\left(\mathbb{1} - \|\boldsymbol{q}\|^{-2}\boldsymbol{q}\boldsymbol{q}^T\right) \end{aligned} \quad (18)$$

1) Position: As in (3), the position is the integral of the velocity in the same frame, all partial derivatives are zero except for the

$$\frac{\partial}{\partial \boldsymbol{v}} \dot{\boldsymbol{p}} = \mathbb{1}. \tag{19}$$

2) Orientation: From (3), we can see that it only depends on the orientation $\boldsymbol{q}$ and the bodyrates $\boldsymbol{\omega}$. From this, we get for the partial derivative with respect to the orientation:

$$\frac{\partial}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \cdot \frac{\mathbb{1} - \|\boldsymbol{q}\|^{-2} \boldsymbol{q}\boldsymbol{q}^T}{\|\boldsymbol{q}\|} \tag{20}$$

and similar for the derivation with respect to bodyrates:

$$\frac{\partial}{\partial \boldsymbol{\omega}} \dot{\boldsymbol{q}} = \frac{1}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & -q_y \\ q_z & q_w & q_x \\ -q_y & q_x & q_w \end{bmatrix}. \tag{21}$$

3) Velocity: From (3), we also get the partial derivatives from the quaternion rotation function and the thrust:

$$\frac{\partial}{\partial \boldsymbol{q}} \dot{\boldsymbol{v}} = 2c \begin{bmatrix} q_y & q_z & q_w & q_x \\ -q_x & -q_w & q_z & q_y \\ q_w & -q_x & -q_y & q_z \end{bmatrix} \cdot \frac{\mathbb{1} - \|\boldsymbol{q}\|^{-2} \boldsymbol{q}\boldsymbol{q}^T}{\|\boldsymbol{q}\|} \tag{22}$$

$$\frac{\partial}{\partial c} \dot{\boldsymbol{v}} = \begin{bmatrix} (q_w q_y + q_x q_z) \\ (q_y q_z - q_w q_x) \\ (q_w^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix}. \tag{23}$$

E. Trajectory Tracking Scheme

To track the reference state along a predefined trajectory, one could use multiple tracking schemes, for example:

- **temporal tracking** where the reference state $\boldsymbol{x}_0(t)$ is chosen based on the passed time,
- **spatial tracking** where the reference state $\boldsymbol{x}_0(\boldsymbol{p})$ closest to the position of the system is chosen.

Since temporal tracking requires the system to follow the trajectory with exact computed timing, it is often less robust against disturbances and model uncertainties. Therefore, we chose spatial tracking, similarly to the approach in [17]. In each control loop, we search on the remaining trajectory for the reference state that minimizes the spatial distance to the actual vehicle position. We do this by applying algorithm 1, where $\boldsymbol{p}_{des,i}$ denotes a position on the trajectory at index i, and $\boldsymbol{x}_{des,i}$ denotes the state at index i. As a result, i is the index of the next reference state on the trajectory, γ is the interpolation value, and $\boldsymbol{x}_{des}$ is our linearly-interpolated closest desired state on the trajectory.

Algorithm 1 for spacial tracking.

```
repeat
    $i \leftarrow i + 1$
    $\boldsymbol{d} \leftarrow \boldsymbol{p}_{des,i} - \boldsymbol{p}_{des,i-1}$
    $\gamma \leftarrow (\boldsymbol{p} - \boldsymbol{p}_{des,i-1}) \cdot \boldsymbol{d} / \|\boldsymbol{d}\|_2^2$
until $\gamma < 1$
$\boldsymbol{x}_des \leftarrow \boldsymbol{x}_{des,i-1} + \gamma(\boldsymbol{x}_{des,i} - \boldsymbol{x}_{des,i-1})$
```

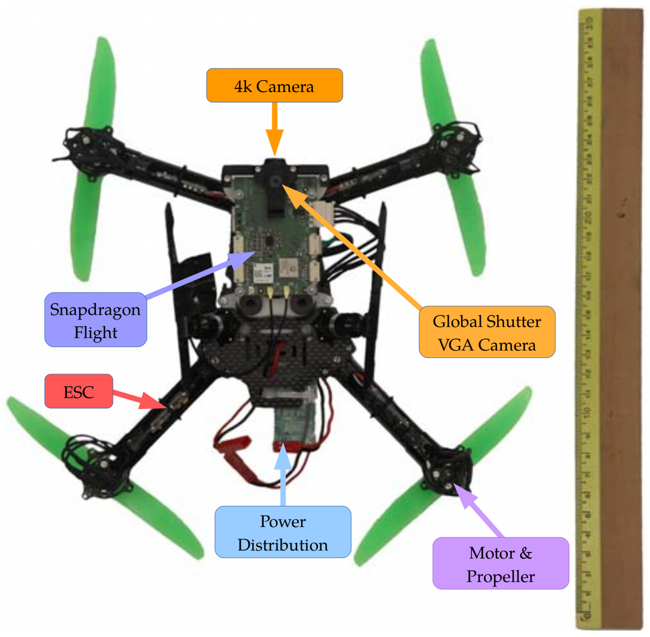

Fig. 3. Our Quadrotor seen from below with a 30 cm ruler. The Snapdragon Flight is visible on top with the global shutter camera. Four "DYS SN20A" ESCs are placed in the arms of the "RKH 250" frame to feed the "DYS BX1306-3100kV" motors with "Gemfan 5030" propellers.

III. Experimental Setup

A. Experiment Platform

Our experiment platform is designed to be lightweight and easy to operate. We achieved this at a total take-off weight of 255 g. The combination of such low inertia and a carbon fiber frame make it extremely robust. It consist of consumer-grade electronic speed controller (ESC), motors, propellers, and a frame with 250 mm diagonal motor distance. Using this components, it can deliver a maximal thrust-to-weight ratio of ~ 2. As flight controller, we use a Snapdragon Flight[1] single-board solution, visible in Fig. 3. It includes a computational unit, IMU, front-facing color camera at a 4k-resolution and down-facing global-shutter gray-scale camera at VGA resolution.

B. Control Architecture

The main software is split into a fast low-level controller for the bodyrates and our superimposed high-level LQR controller, as already described in Sec II-A. This architecture is illustrated in Fig. 2. On the Snapdragon Flight's Application Digital Signal Processor (aDSP); the low-level controller runs in the form of the PX4[2] flight stack. This controller reads the IMU onboard the Snapdragon Flight with 8 kHz and computes the PWM signal for the ESC from the desired bodyrates and collective thrust.

The high-level controller runs on the main processor (2.26 GHz quad-core ARM) using Linaro Linux[3] provided

[1]Qualcomm: developer.qualcomm.com/software/machine-vision-sdk
[2]PX4 flight stack: www.px4.io
[3]Linaro: www.linaro.org

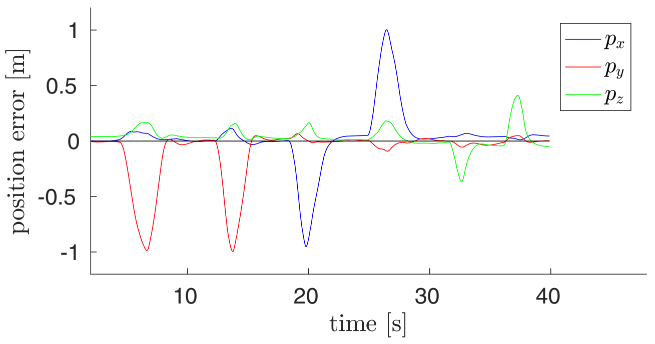

Fig. 4. Position error during a hover task. The vehicle was disturbed by pulling it away from its reference point as seen in the accompanying video.

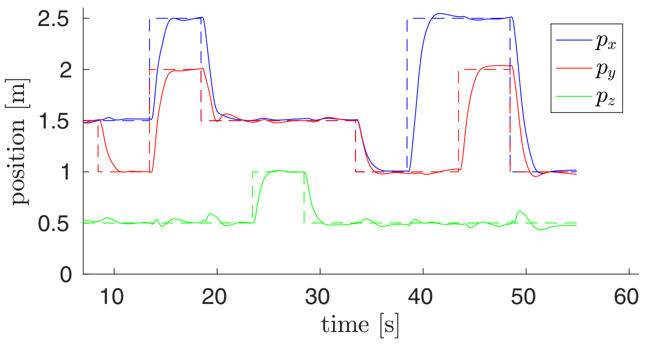

Fig. 5. Position during step changes in the reference all 5 seconds. The dashed line marks the reference, while the full line marks the estimate.

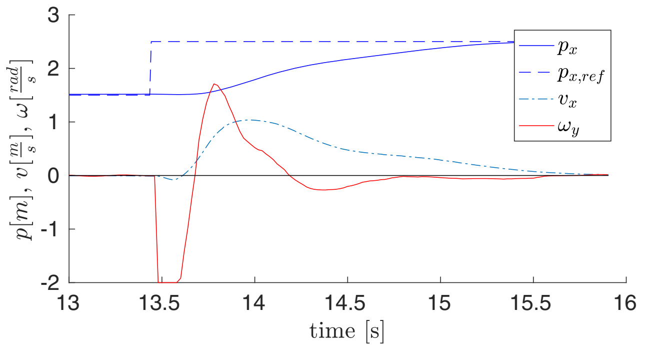

Fig. 6. Input ω_y with reference $p_{x,ref}$, position p_x and velocity v_x during step changes in the reference.

by Intrinsyc[4] with ROS[5]. To get an accurate state estimate, we use a Visual Inertial Odometry (VIO) by Qualcomm[1]. The IMU readings are down-sampled and forwarded to the VIO pipeline at 250 Hz together with a VGA gray-scale image from the down-facing global-shutter camera at 30 Hz. The pose estimate after each camera frame is fed back to the PX4 flight stack, which fuses it together with IMU signals to get a low drift, high frequency state estimate at 250 Hz.

Our proposed LQR controller uses this estimate to compute the target bodyrates and collective thrust at 100 Hz. Furthermore, it recalculates the cost-optimal gain matrix $\boldsymbol{K}$ from (15) at 10 Hz using the linearization of the system at the most recent estimated state. To chose the tracking reference, we use the the spatial tracking scheme from Sec. II-E due to better handling of model uncertainties and disturbances. All experiments were done with the quadratic cost terms set to $\boldsymbol{Q} = diag([100, 100, 100, 1, 1, 1, 1, 10, 10, 10])$ and $\boldsymbol{R} = diag([1, 5, 5, 0.1])$, while the inputs were limited at $\|_B\boldsymbol{\omega}_{des}\| \leq 2\,\mathrm{rad\,s^{-1}}$ and $2\,\mathrm{m\,s^{-2}} \leq_B c_{des} \leq 18\,\mathrm{m\,s^{-2}}$ for all experiments.

C. Experiment Description

We perform four different experiments to show basic stability of our controller: hover with reference jumps and disturbances (III-C.1); tracking a feasible minimum-snap trajectory (III-C.2); tracking an infeasible trajectory (III-C.3); disturbing the vehicle during tracking a feasible trajectory (III-C.4). For all experiments, we use onboard VIO only, without any external motion capture system. Because the floor and walls provide practically no texture, we place some textured carpets in the room.

1) Hover with Reference Jumps and Disturbances: In this experiment we let the vehicle hover and apply step-changes to the reference and disturbances by pulling the vehicle from its hover position. While the reference is momentarily constant, we still apply the state-dependent LQR to adjust to the estimated vehicle state. No trajectory or feasibility checks are needed, since the pure state-feedback is enough to stabilize the vehicle around and towards the reference.

[4]Intrinsic: www.intrinsyc.com
[5]Robot Operating System: www.ros.org

2) Tracking a Feasible Trajectory: We generate a feasible trajectory using the minimum-snap approach from [2]. For this, we set four waypoints at positions $\boldsymbol{p}_{ms0} = [0, -1.2, 0]^T$, $\boldsymbol{p}_{ms1} = [1.5, 0, 0.5]^T$, $\boldsymbol{p}_{ms2} = [0, 1.5, 0]^T$, $\boldsymbol{p}_{ms3} = [-1.5, 0, 0.5]^T$ and compute a minimum-snap trajectory sampled with $dt = 0.1\,\mathrm{s}$. We limit the velocity to $3.0\,\mathrm{m\,s^{-1}}$. This results in a trajectory close to an ellipse, except for the start and end phase.

3) Tracking an Infeasible Trajectory: In this experiment, we show the stability of the controller with respect to infeasible references. We generate a perfectly elliptical trajectory, parametrized with respect to the angle α by $\boldsymbol{p}_{ell}(\alpha) = [r_x \sin(\alpha), -r_y \cos(\alpha), r_z \sin(\alpha)]^T$ with $r_x = 1.5\,\mathrm{m}$, $r_y = 1.2\,\mathrm{m}$, $r_z = 0.5\,\mathrm{m}$ and a sample with $d\alpha = 2\pi/36$. The velocity is set to point from one reference point to the next one with a magnitude of $2.0\,\mathrm{m\,s^{-1}}$. The reference orientation is perfectly horizontal as in hover. Note that, therefore, the velocity is discontinuous, the acceleration is infeasible, and no feed-forward part exists.

4) Tracking with Disturbance: We generate a feasible trajectory as in III-C.2 but with a lower maximal velocity of $1.0\,\mathrm{m\,s^{-1}}$. During execution, we stop the vehicle by holding or pulling it away from the trajectory. Because of the spatial tracking, it continues along the trajectory without skipping any segments, which instead could happen in temporal tracking as mentioned earlier (II-E).

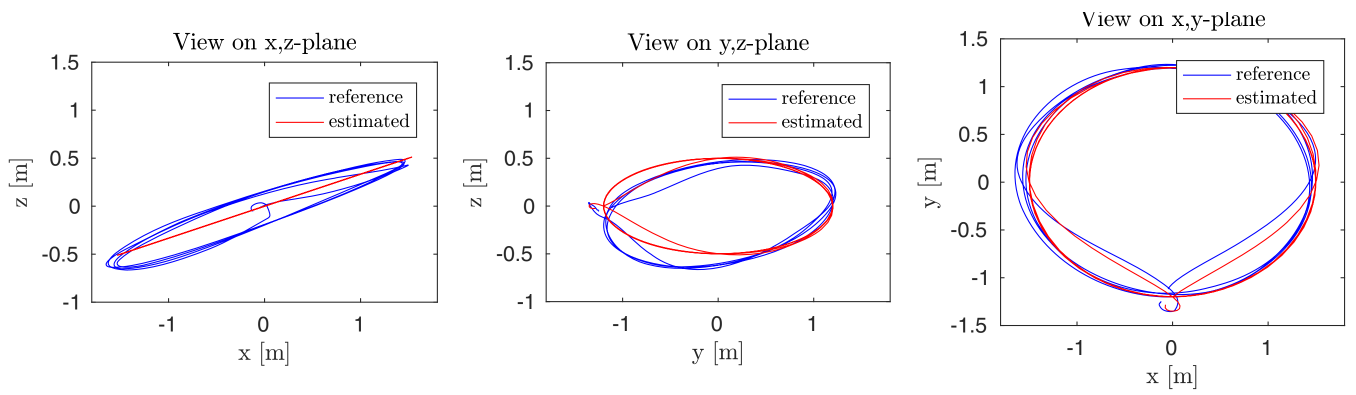

Fig. 7. View from each world axis on a feasible minimum-snap trajectory with $v_{max} = 3.0\,\mathrm{m\,s^{-1}}$ executed using only VIO and our state-dependent LQR. Note the error in z-direction due to thrust uncertainties.

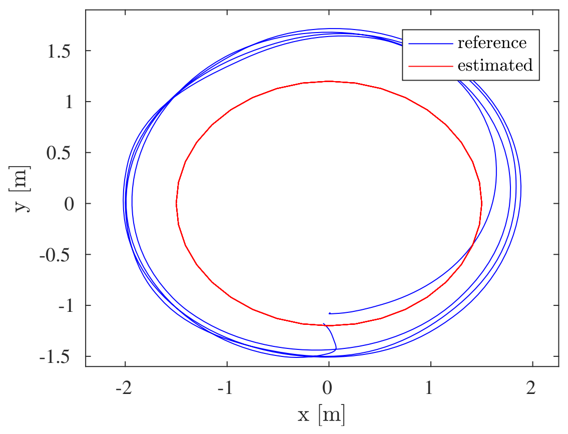

Fig. 8. Stabilizing the vehicle along an unfeasible trajectory. Note the deviation outward of the ellipse due to unfeasible velocity and attitude reference, resulting in a error of $\sim$0.5 m.

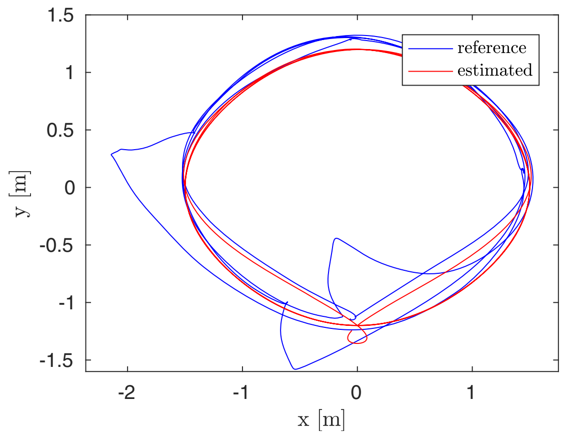

Fig. 9. Trajectory tracked with a human operator disturbing (pulling) the vehicle. Note the convergence back to the trajectory without skipping segments due to spatial tracking.

IV. Results

All experiments are visible in the accompanying video at: `https://youtu.be/8OVsJNgNfa0`

A. Hover with Reference Jumps and Disturbances

First, we note the non-zero static error in position tracking, which results from model uncertainties and the lack of an integrative part in the controller. In Fig. 4, we can see the position error due to a large disturbance by a human. Note how the disturbance is rejected with very little overshoot. Another property is the bounded velocity due to position- and velocity-error saturation. In Fig. 5, we depict the reaction to jumps in the reference. Note how small deviations from the z-axis setpoint are caused by large jumps in x and y as a trade-off to lower the quadratic state cost as fast as possible. Additionally, we show the an input in Fig. 6 during a reference jump. Note that the input jumps due to the instantaneous jump in the reference, but otherwise is smooth even over the updates of the LQR gains at 10 Hz.

B. Tracking a Feasible Trajectory

While tracking a feasible trajectory introduces no more complexity than our hover experiment except for a moving reference state, we expect the same results as in IV-A. In fact, we can again observe some small offset from the reference trajectory, as depicted in Fig. 7, resulting from model uncertainties. The biggest influence is the uncertainty of the collective thrust, since this can change with battery voltage and is not accounted for in the low level controller, resulting in the observable deviation in z-direction (calibrated to minimal z-error in hover at 70% charging state). We can observe this deviations in the first and second plot in Fig. 7, as well as the non symmetry of it due to gravity. This results in a too low position when moving upward, and a too high position when moving downward. The last plot of Fig. 7 shows the x, y-plane depicting the horizontal tracking performance with little position error.

C. Tracking an Infeasible Trajectory

With this experiment, we want to show stability of the controller when the reference is infeasible. We can see in Fig. 8 how the vehicle deviates outward from the reference trajectory due to the lack of a correct reference attitude and infeasible velocity changes. Still, the controller manages to follow the trajectory and stabilize the vehicle, with a trade-off in positional accuracy.

D. Tracking with Disturbance

In our last experiment, we show how the spatial tracking performs under significant disturbances during execution, depicted in Fig. 9. While a time-based tracker would not account for the disturbance and possibly skip segments after a disturbance, our spatial tracker is not influence by catching, holding and pulling the vehicle from the trajectory. The algorithm for spatial tracking simply keeps the reference at the last sample point while we hold the vehicle back. As soon as we release the vehicle, it converges back onto the trajectory and continues tracking without skipping any segments, as visible in our video. Note that this adds to the robustness of the implementation and covers many failure scenarios introduced by disturbance as well as model uncertainties.

E. Computation Time

Our algorithm computes the solution for the state-dependent LQR at $10\,\mathrm{Hz}$ with a mean computation time of $\bar{t}_c = 36\,\mathrm{ms}$ with a standard deviation of $\sigma_c = 11\,\mathrm{ms}$ in an average of $\bar{n}_i = 41$ iterations.

V. Discussion

In this work, we showed the fundamental applicability of state-dependent and, therefore, time-varying LQR control on a MAV with real experiments, where other approaches only show LQR control for attitude (linearized around one reference state) or only deliver simulation results. While we did not directly compare to other approaches, we intended to mitigate problems of hierarchical control schemes and show the feasibility of the state-dependent LQR approach using our implementation, proven by our experiments. Two topics remain up for discussion: (V-A) optimality under uncertainties due to state estimation and (V-B) extensions of our approach.

A. Separation Theorem

Since modern MAVs use visual integral state estimation as in our approach, it is interesting to discuss the stability of our controller in conjunction with such an estimator. Many of these estimators are based on a Kalman filter as in [18], [19]. Due to the separation principle, the combination of such an estimator and an LQR will still be stable and optimal if both individual components are stable. Therefore, this approach is valid for a wide variety of state estimation pipelines.

B. Extension of our Approach

The true advantage of our controller and linearization is its possible extensions, such as: (I) integral states to counteract steady-state errors; (II) implementation of an actuator dynamics model to cover uncertainties and actuator limitations; (III) development of a iterative LQR for planning and model predictive control; (IV) predictive control for cost-field-based obstacle avoidance; (V) extension to other platforms. We will build upon our findings and investigate the implementation of these extensions to develop a more elaborate control pipeline.

VI. Conclusion

We proposed a state-dependent LQR which couples the translational and rotational states of the vehicle, and provides the dynamics and linearization needed. We implemented this approach on a lightweight quadrotor using a low-power Qualcomm ARM platform and run all state estimation, LQR computation and controller onboard. Our implementation is efficient enough to update the LQR gains at $10\,\mathrm{Hz}$ and our experiments prove feasibility and robustness with this control approach.

References

[1] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, “Aggressive quadrotor flight through narrow gaps with onboard sensing and computing,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017.

[2] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2011, pp. 2520–2525.

[3] M. Mueller, S. Lupashin, and R. D’Andrea, “Quadrocopter ball juggling,” in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2011, pp. 4972–4978.

[4] S. Tang and V. Kumar, “Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2015, pp. 2216–2222.

[5] P. Foehn, D. Falanga, N. Kuppuswamy, R. Tedrake, and D. Scaramuzza, “Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload,” in *Robotics: Science and Systems (RSS)*, 2017.

[6] R. E. Bellman, “The theory of dynamic programming,” *Bull. Amer. Math. Soc.*, 1954.

[7] R. Bellman, *Dynamic Programming*, ser. Dover Books on Computer Science Series. Dover Publications, 1957.

[8] R. Kalman, “A new approach to linear filtering and prediction problems,” *J. Basic Eng.*, vol. 82, pp. 35–45, 1960.

[9] D. Mellinger, N. Michael, and V. Kumar, “Trajectory generation and control for precise aggressive maneuvers with quadrotors,” in *Int. Symp. Experimental Robotics (ISER)*, Dec 2010.

[10] M. Hehn and R. D’Andrea, “Quadrocopter trajectory generation and control,” in *IFAC World Congress*, vol. 18, no. 1, 2011, pp. 1485–1491.

[11] T. Lee, M. Leoky, and N. McClamroch, “Geometric tracking control of a quadrotor uav on se(3),” in *IEEE Conf. Decision Control (CDC)*, Dec. 2010, pp. 5420–5425.

[12] A. Al-Tamimi, F. Lewis, and M. Abu-Khalaf, “Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 943–949, Aug. 2008.

[13] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, and J. Guichard, “LQR control for a quadrotor using unit quaternions: Modeling and simulation,” in *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, Mar. 2013.

[14] P. Reist and R. Tedrake, “Simulation-based LQR-trees with input and state constraints,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2010, pp. 5504–5510.

[15] G. V. Raffo, M. G. Ortega, and F. R. Rubio, “Mpc with nonlinear $\mathcal{H}\infty$ control for path tracking of a quad-rotor helicopter,” *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 8564–8569, 2008.

[16] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart, “Fast nonlinear model predictive control for multicopter attitude tracking on SO(3),” in *IEEE Conf. Control Appl. (CCA)*, Sept. 2015, pp. 1160–1166.

[17] G. Antonelli, E. Cataldi, F. Arrichiello, P. R. Giordano, S. Chiaverini, and A. Franchi, “Adaptive trajectory tracking for quadrotor MAVs in presence of parameter uncertainties and external disturbances,” *IEEE Transactions on Control Systems Technology*, pp. 1–7, 2017.

[18] A. I. Mourikis and S. I. Roumeliotis, “A multi-state constraint Kalman filter for vision-aided inertial navigation,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, Apr. 2007, pp. 3565–3572.

[19] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, “Autonomous, vision-based flight and live dense 3D mapping with a quadrotor MAV,” *J. Field Robot.*, vol. 33, no. 4, pp. 431–450, 2016.