Anytime Computation and Control for Autonomous Systems

Yash Vardhan Pant, Houssam Abbas, Kartik Mohta, Rhudii A. Quaye, Truong X. Nghiem, Joseph Devietti, and Rahul Mangharam

Abstract—**The correct and timely completion of the sensing and action loop is of utmost importance in safety critical autonomous systems. Crucial to the performance of this feedback control loop are the computation time and accuracy of the estimator which produces state estimates used by the controller. These state estimators often use computationally expensive perception algorithms like visual feature tracking. With on-board computers on autonomous robots being computationally limited, the computation time of such an estimation algorithm can at times be high enough to result in poor control performance. We develop a framework for codesign of anytime estimation and robust control algorithms, taking into account computation delays and estimation inaccuracies. This is achieved by constructing an anytime estimator from an off-the-shelf perception-based estimation algorithm and obtaining a trade-off curve for its computation time versus estimation error. This is used in the design of a robust predictive control algorithm that at run-time decides a contract, or operation mode, for the estimator in addition to controlling the dynamical system to meet its control objectives at a reduced computation energy cost. This codesign provides a mechanism through which the controller can use the tradeoff curve to reduce estimation delay at the cost of higher inaccuracy, while guaranteeing satisfaction of control objectives. Experiments on a hexrotor platform running a visual-based algorithm for state estimation show how our method results in up to a 10% improvement in control performance while simultaneously saving 5%–6% in computation energy as compared to a method that does not leverage the codesign.**

Index Terms—**Adaptive systems, autonomous systems, computer vision (CV), predictive control, robust control.**

Manuscript received April 22, 2019; revised November 27, 2019; accepted February 9, 2020. Manuscript received in final form March 3, 2020. This work was supported in part by the U.S. Department of the Transportation University Transportation Center (Mobility21). Recommended by Associate Editor V. Zavala. *(Corresponding author: Yash Vardhan Pant.)*

Yash Vardhan Pant, Kartik Mohta, Rhudii A. Quaye, and Rahul Mangharam are with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: yashpant@seas.upenn.edu; kmohta@seas.upenn.edu; quayerhu@seas.upenn.edu; rahulm@seas.upenn.edu).

Houssam Abbas was with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA. He is now with the Department of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: houssam.abbas@oregonstate.edu).

Truong X. Nghiem was with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA. He is now with the School of Informatics, Computing, and Cyber Systems, Northern Arizona University, Flagstaff, AZ 86011 USA (e-mail: truong.nghiem@nau.edu).

Joseph Devietti is with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: devietti@cis.upenn.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCST.2020.2979388

I. Introduction

THE real-time control of many autonomous robots, for example, self-driving cars and unmanned aerial systems (UASs), usually includes closed loops between the controller that drives the actuation, and the estimator that computes state estimates which are used by the controller. Of particular importance in this traditional feedback control architecture are: 1) the delay in the control action due to the time taken by the estimator for computing the state estimate and 2) the inaccuracy in the state estimate. Either of these factors can result in control actions that can drive the system into an unsafe state that it should not reach, for example, a no-fly zone.

In most conventional feedback control designs, controllers are tasked with realizing the functional goals of the system under simplistic assumptions on the performance of the estimator, in particular, perfect state estimates and negligible computation time. This design principle based on the separation of concerns simplifies the control design process but often does not accurately reflect real implementations. On the other hand, most perception-based state estimation algorithms [1], [2] do not take into account how their output will be used to close the control loop. More specifically, an estimator will more often than not *run to completion*: that is, its termination criteria are designed to provide the best quality output (estimate). This can result in large delays in the control action, leading to degraded control performance. It can also result in the computation platform consuming a significant amount of energy, reducing the amount of time the system can operate on a full charge. This is especially of concern in mobile robotic systems like autonomous drones and cars that operate on batteries with limited capacity.

In this article, we focus on these problems, showing that when the real-time requirements on the closed-loop system become more demanding, this disconnect between the estimator and controller can lead to poor system performance. The following example shows how this problem can manifest in even simple settings.

Example 1: To illustrate the impact of estimation delay δ and state estimation error ϵ on control performance,[1] we show a simple PID tracker controlling the motion of a point mass in the (x, y)-plane. The position of the point mass must follow a reference constant trajectory, the x-dimension of which is shown in Fig. 1 (the same plot can be obtained for the y-position). We simulate two cases of estimation (and therefore actuation) delay and error, where a larger delay

[1]In this article, we judge the control performance by a cost function that accumulates the tracking error and control effort over time [see (8)]. In general, the lower the cost, the better the control performance.

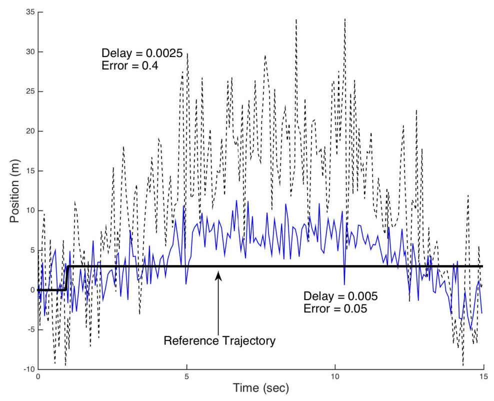


Fig. 1. Effect of delay, error values on control performance of a PID tracker.

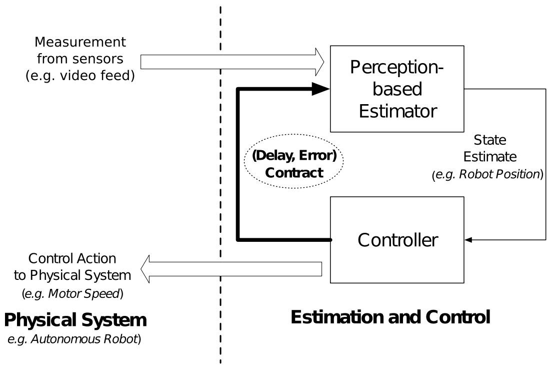

Measurement from sensors (e.g. video feed)
Perception-based Estimator
(Delay, Error) Contract
State Estimate (e.g. Robot Position)
Controller
Control Action to Physical System (e.g. Motor Speed)
Physical System
e.g. Autonomous Robot)
Estimation and Control

Fig. 2. Contract-driven controller and estimator.

value δ implies a smaller estimation error ϵ. As can be noted in Fig. 1, the effect of delay can be nonnegligible. In this example, it can be seen that the increased delay causes the tracking performance to worsen. Running an estimation task with a fixed smaller delay but larger estimation error does not necessarily solve the problem of degraded performance, as can be seen in Fig. 1. Therefore, there is a need to rigorously quantify the tradeoff between computation time and estimation error and then exploit that tradeoff to achieve the best control performance under the problem constraints. Rather than always running the estimation task to completion, it is useful to have several delay/error run-time modes for the estimator. These can then be used at run-time to satisfy the control objectives. □

The goal of this article is to develop a rigorous framework for the codesign of the controller and estimation algorithms. Here, the estimator has a range of computation time/estimate quality operating modes, and in order to best maintain control performance and reduce energy consumption, the controller at run-time selects one of these modes for the estimator to operate in. This is motivated by the following observations.

1) The traditional engineering approach to account for the estimator's run-time is to gauge the worst case execution time (WCET) [3] of the estimation task, and design the system to meet deadlines under the WCET conditions. In practice, however, the actual execution time of perception-based estimators can be much less than the WCET and depends on the actual data being processed. Hence, considering the WCET can lead to a conservative design of the system. Additionally, the classical timing analysis alone does not guarantee *functional* correctness of the closed-loop system under control.
2) Moreover, in the context of closed-loop control, we do not always require the best quality state estimate: more often than not, a lower quality estimate, computed using lesser energy and time, is acceptable to achieve the control objectives.
3) In the case where obtaining a better quality state estimate requires longer computation time, it can be detrimental to the control performance to require a high-quality state estimate all the time. For example, when the onboard computer is overloaded, there may be a need to spend less time computing a state estimate so that not only the control action has less delay, but also so that other processes can access the computation resource as scheduled.

In this article, we develop the observations above into a *codesign framework* for real-time control systems, where the controller and estimator are interfaced via *contracts*. A contract is an assurance requested by the controller, and provided by the estimator, that the latter can give an estimate with a certain accuracy ϵ, and within a predefined time deadline δ. The computation time is given to the estimator, as well as the quality of the state estimate, which defines the contract. This can be interpreted as turning the estimator into a discretized version of an *anytime algorithm* [4] where its computation can be interrupted at runtime to get a state estimate, usually with a trade-off between the computation time given to the algorithm and the quality of output that it returns. Through this notion of contracts, we show how the controller can vary the computation time of the estimation algorithm to maintain control performance and to reduce energy consumption. The work presented here is focused on estimation algorithms that rely on computationally intensive computer vision (CV) algorithms in order to get a state estimate of a dynamical system, for example, those in autonomous robot navigation with visual (camera, LiDAR) sensors. We refer to these as *perception-based estimators*. Through experiments, we show that the computation time of such algorithms can be significant (and much greater than that of the control algorithm), resulting in an adverse impact on the closed-loop control performance.

The architecture for the codesign framework proposed in this article is shown in Fig. 2. It resembles the conventional closed-loop control architecture involving the estimator, the controller, and the system being controlled, but also incorporates the (delay, error) contract as an interface between the controller and the estimation algorithm.

Summary of Contributions: In this article, we build upon our results from the work of Pant *et al.* [5] and present a framework for the codesign of control and estimation algorithms for the real-time control of dynamical systems. This approach consists of the following.

1) A well-defined interface between control and estimation, in the form of operating modes, or *contracts*, on the accuracy and computation time of the estimator (Section III).
2) Characterizing the estimator accuracy as either deterministic (worst case) or stochastic through offline profiling of the perception-based estimator (Section VII).
3) A predictive control algorithm that can change the operating mode of the estimator at run-time to achieve control objectives at a lower energy cost (Section IV), while providing guarantees on the satisfaction of constraints

for both deterministic (Section V) and probabilistic (Section VI) characterizations of the estimation error.

4) A straightforward, low-touch and low-effort approach to design a contract-driven estimation algorithm starting from an off-the-shelf, run-to-completion version of it (Section VII).
5) We demonstrate our method on an autonomous flying robot and show its performance and energy gains over a classical controller (Section VIII).

Compared to our previous work [5], which only allows for characterizing the contracts in terms of worst case estimation error, in this article, we extend the framework to also allow for a probabilistic representation for the estimation error. We also provide a guarantee on the satisfaction of constraints and recursive feasibility of the new control predictive algorithm resulting from this probabilistic setup. In addition, we also extend the experimental setup and incorporate a real-time implementation of the new control algorithm and evaluate our approaches with two sets of new experiments on a hexrotor autonomous robot.

II. Related Work

Algorithms that can be interrupted at any point at run-time and still return an acceptable solution are called *anytime algorithms* [4]. Such algorithms generally return solutions with improving the quality of output the longer they run for. A subset of these is contract algorithms [6] which can be interrupted only at a finite number of preagreed-upon times. In this article, we design a contract-driven perception-based state estimator but significantly expand the notion of a contract to now include the *quality of the solution* (estimation error in our case) as well as the computation time.

Anytime algorithms have found particular importance in the field of graph search [7], evaluation of belief networks [8], and GPU architectures [9], [10]. With autonomous systems gaining popularity, computationally overloaded systems with real-time requirements are becoming the norm. This has generated interest in the development of any time algorithms in the field of control theory, with Quevedo and Gupta [11], Bhattacharya and Balas [12], and Fontanelli *et al.* [13] exploring this line of research. Anytime algorithms have also found widespread use in the field of motion planning [14]–[17].

The work presented in this article contrasts considerably with these efforts as the assumption of anytime computation is not on the controller or planning side but on the perception-based state estimation component of the feedback control loop. The loop is closed by the control algorithm presented here that decides the contract for the anytime state estimator at run-time. Also different from the works discussed above, which require instantaneous and perfect full state access for the controller, our control algorithm takes into account the computation time and the estimation error of the perception-based estimators that are common in autonomous systems. The recent work of Falanga *et al.* [18] also tackles the problem of codesigning the perception and the control but does so as a joint optimization that takes into account both the perception and the control. This article differs from the work of Falanga *et al.* significantly as we introduce the notion of contracts to decouple the perception-based estimator's performance and control optimization. Our method also explicitly incorporates the timing and the estimation performance of the perception-based estimator in the control design, and can be used for the off-the-shelf perception-based estimators (e.g., Section VII-C). Although the control algorithms developed in this article are limited to linear time-invariant (LTI) systems, recent work in robust predictive control for nonlinear systems that can be feedback linearized [19] suggests that a framework similar to ours can also be applied to such nonlinear systems. This is, however, beyond the scope of this article and is left for future exploration.

In the domain of real-time systems, worst case analysis, along with logical execution time semantics, are used in [20] to imbue a controller with information of the timing characteristics of the closed-loop implementation. On the other hand, our approach involves profiling the estimation algorithm in a direct manner to get timing and estimation error characteristics. Although [20] involves formally verifying a given controller, we *design* a control algorithm that is correct by construction and takes advantage of delay/accuracy tradeoffs in real-time. In the context of autonomous multirotor UAVs, the effect of increasing the computation time of task on the overall performance of the system has been analyzed in [21] by using a resource allocation algorithm similar to QRAM [22]. Our approach contrasts with this as we focus on the execution time of a particular task, the perception-based state estimator, which directly impacts the closed loop control performance. In addition to this, we also formulate a controller that provides mathematical guarantees on the system's performance.

Finally, in the area of computer architecture, approximate computing approaches [23]–[25] have been explored to get savings in time or energy by performing a computation in an approximate manner, rather than precisely. Although anytime algorithms and approximate computing have a common high-level goal, approximate computing methods are run-to-completion and lack a feedback mechanism to permit computation and resources to be balanced dynamically. Also, the time and energy scale that our approach deals with are much greater than those that concern approximate computing.

III. Codesign of Estimation and Control

Conventional closed-loop control systems are generally designed in a manner where the controller is incognizant of the implementation details of the state estimation module, whereas the estimation module is designed independent of the requirements of the controller. For example, a feedback controller, that gets state estimates from a camera-based visual odometry algorithm, might not be designed to take into account the nonnegligible time taken to process the video frames to get a state estimate. We refer to this computation time as the estimation delay. On the other hand, the design of most perception-based estimators does not take into account the varying real-time constraints that the controlled closed-loop system must satisfy. Also of importance, especially in autonomous systems deployed in the field, is the power consumed by the computation platform which can have a significant impact on the duration the system can operate between charging.

Taking these factors into account, we propose the *codesign* of estimation and control to improve the closed-loop performance of real-time control on systems with computationally and power-limited platforms. This is done through a *contract-driven framework* for both estimator and controller in which the controller asks for a state estimate within a certain deadline δ seconds, with an associated bound on the inaccuracy of the estimation. This inaccuracy can either be in the form of a hardbound ϵ, for example, an infinite-norm bound on the estimation error vector, or have a probabilistic characterization Σ, for example, the covariance of the estimation error vector, depending on the application. For the sake of simplicity, we use ϵ for the characterization of the estimation error in the following text.

In our framework, the tuple (δ, ϵ) forms the *contract* between controller and estimator. The estimator is tasked with providing a state estimate that respects the contract. Aware of these contracts, the controller can set the appropriate contract in a time-varying manner to adapt the closed-loop system performance in real-time to take into account the control requirements of the physical system. For example, it can decide when an estimate is needed fast (but usually with higher error), and when a more accurate estimate is needed (but with greater delay). Note, the (δ, ϵ) contract can also be thought of as setting an *operating mode* for the perception-based estimator. A high-level view of this setup is shown in Fig. 2.

In order to make sure that the contracts are such that the estimator can indeed fulfill them, the estimator is profiled offline. To do this, the estimator's internal parameters are varied, and for each parameter setting, it is run on a *profiling data set* (with a known ground-truth baseline). This results in a set of (δ, ϵ) values, each one corresponding to a particular setting of the parameters. These values can be plotted on a curve, which we call the *error-delay curve* made up of discrete points, (δ, ϵ). Examples of such a curve are shown in Figs. 6 and 8. Section VII provides the detailed procedure for obtaining this curve for a perception-based estimator.

During run-time execution, upon receiving a (δ, ϵ) contract request from the controller, the estimator can adapt its parameter settings to fulfill the contract, that is, to provide a state estimate within the requested deadline δ that also respects the requested error bound ϵ.

The controller, in the codesign framework, is designed with the awareness of the error-delay curve of the estimation algorithm, and requests contracts from that curve. The error-delay curve, thus constitutes the *interface* between the controller and state estimator. The controller leverages the flexible nature of the estimation algorithm to maximize control performance.

The closed-loop architecture in a system with codesign of the estimator and controller is shown in Fig. 3. In this codesigned system, the controller can make the estimation algorithm switch to lower or higher time (and/or energy) consuming modes based on the control objective at the current time step. The main components of the codesign architecture presented in this article are: 1) a contract perception-based estimator; 2) a robust control algorithm that computes an input to be sent to the physical system being controlled as well as the contract for the estimator; and 3) the interface between them. More details are presented in this article.

IV. Control With Contract-Driven Estimation

In this section, we formalize how the control algorithm utilizes the error-delay curve of the estimator to optimize control performance while minimizing the power consumed by the computations for the perception-based estimator.

A. System Model

In order to model the codesign process, consider the closed-loop control of an autonomous hexrotor robot

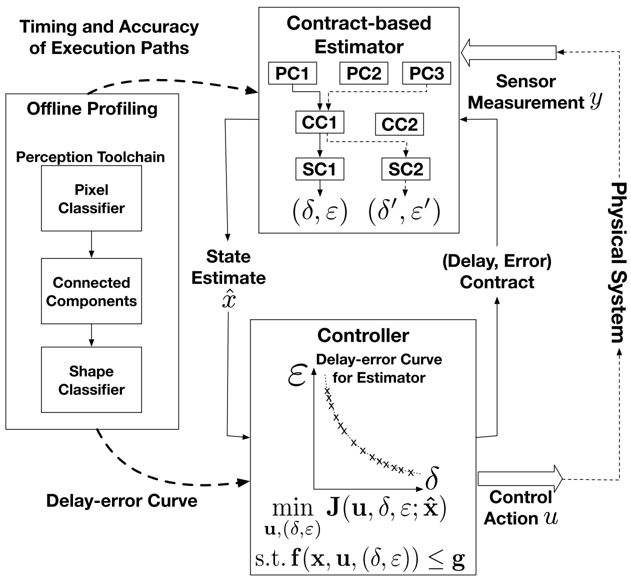


Fig. 3. Contract-driven estimator and controller. With knowledge of the estimator's performance through offline profiling, the controller both actuates the dynamical system and sets *contracts* for the estimator at run-time in order to maximize control performance while guaranteeing that constraints on the system are always satisfied.

(more details in Section VIII). The state x of the hexrotor consists of its 3-D position and 3-D velocity, whereas the input u to the robot consists of the desired pitch and roll angles, and the desired thrust. The hexrotor's task is to fly a predefined trajectory given by x_{ref}, where $x_{ref}(t)$ gives the desired position at each time t. The dynamics of the hexrotor, relating the time-evolution of its state to the current state and input, can be linearized around hover and approximated by the following LTI ODE:

$$\dot{x}(t) = A_c x(t) + B_c u(t) + w_c(t). \tag{1}$$

Here, the state vector $x \in \mathbb{R}^n$ is constrained to be within set $X \subset \mathbb{R}^n$, the control input $u \in \mathbb{R}^m$ is constrained within set $U \subset \mathbb{R}^m$, and $w_c \in \mathbb{R}^n$ is the process noise assumed to lie in a (bounded) set $\mathcal{W}_c \subset \mathbb{R}^n$. $A_c \in \mathbb{R}^{n\times n}$ and $B_c \in \mathbb{R}^{n\times m}$ are matrices. LTI ODEs can model a wide range of systems, and our results apply to arbitrary LTI systems of the form given in (1) with compact and convex constraint sets X, U and $\mathcal{W}_c$. The sets X and U are determined by the control designer or by physical constraints on the system. For example, X captures limits on the state to define the region in which the hexrotor can fly and the velocity limits on it. The set U restricts the inputs to values that can be supported by the rotors, as well as within which the linearized system provides a good approximation to the true nonlinear dynamics.

B. Time-Triggered Sensing and Actuation

For feedback control of the hexrotor, the controller needs to be aware of the hexrotor's current position and speed, that is, requires an *estimate* of its current state x. This is done via a perception-based estimator, that processes video frames (at a fixed rate) obtained through a downward-facing camera mounted on the hexrotor. The estimator detects and tracks features across frames, and deduces its own position through the relative motion of these features.

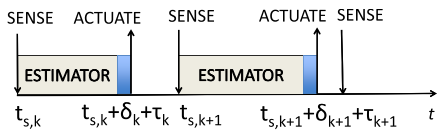


Fig. 4. Time-triggered sensing and actuation. The figure shows the varying execution time for the estimator and the blue area shows the execution time for the controller, which is small.

A new frame is captured by the camera every $T > 0$ seconds, which results in periodic measurements at instants $t_{s,k} = kT$, where $k \in \mathbb{N}$. This measurement is used by the estimator to compute the state estimate $\hat{x}_k := \hat{x}(t_{s,k})$ with the desired accuracy ϵ_k *determined by the contract set by the controller in the previous time step*. The controller then acts on this state estimate to compute the control input u_k as well as decide on the perception-based estimator's delay and accuracy contract $(\delta_{k+1}, \epsilon_{k+1})$ for the next time step. The control is then applied to the physical system according to (1) at instant $t_{a,k} = t_{s,k} + \delta_k + \tau_k$, where τ_k is the time it takes to compute the input. See Fig. 4 for the timing diagram of this process.

The controller has access to the delay-error curve, or operating modes Δ of the estimator, and at each time step selects contracts *from that curve*. This curve is obtained offline as explained in Section III, and illustrated in Section VII. Note that at each step $k \geq 0$, the estimation accuracy ϵ_k, and hence the delay δ_k are already decided in the previous time step and known to the controller. For the very first step $k = 0$, the initial estimation mode δ_0, ϵ_0, as well as the initial control input u_{-1} are chosen by the designer.

C. Control Performance

The controller has a goal that is twofold: it needs to ensure that the reference trajectory is tracked as closely as possible, and that the computation energy consumed to do so is minimized. To capture this, we define two (stage) cost functions: first, $\ell(x,u) = (x - x_{ref})^T Q (x - x_{ref}) + u^T R u$ defines a weighted sum of the tracking error (first summand) and the input power (second summand). Here, Q and R are positive-semidefinite and positive-definite matrices, respectively. Second, $\pi(\delta)$ captures the average power consumed to perform a perception-based estimation computation duration δ. This power information is collected offline during the estimator profiling phase.

The total cost function for the controller to minimize is $J = \sum_{k=0}^{M} (\ell(x_k, u_k) + \alpha \pi(\delta_k))$, where $M \geq 0$ is the duration of the system's operation.

D. Discretized Dynamics

Due to the time-triggered sensing and actuation of the system (see Section IV-B), from time $t_{s,k}$ to $t_{a,k}$, the previous control input u_{k-1} is still being applied. Then at $t_{a,k}$ the new control input u_k is computed and applied by the controller (see Fig. 4). For the sake of simplicity, we assume the computation time for the controller (τ) is small and constant, and so lump it with the time for the estimator (δ). This is justified experimentally for our problem (in Section VIII) where the time for the controller is negligible compared to the time taken by the estimation algorithm. The discrete time dynamics for this setup, with a periodic sensing time of T, are given by

$$x_{k+1} = A x_k + B_1(\delta_k) u_{k-1} + B_2(\delta_k) u_k + w_k \quad k \geq 0 \tag{2}$$

in which

$$A = e^{A_c T}, \quad w_k = \int_0^T e^{A_c (T-t)} w_c\left(t_{s,k} + t\right) dt$$

$$B_1(\delta) = \int_0^\delta e^{A_c(T-t)} B_c dt, \quad B_2(\delta) = \int_\delta^T e^{A_c(T-t)} B_c dt.$$

Here, w_k is the process noise accumulated during the interval. It is constrained to lie in a compact convex set $\mathcal{W}$ since $w_c(t)$ lies in the compact convex set $\mathcal{W}_c$ and T is finite. As explained above, both the current control u_k and the previous control u_{k-1} appear in (2). In addition, the input matrices $B_1(\delta_k)$ and $B_2(\delta_k)$ depend on the delay δ_k. The estimation accuracy ϵ_k, indirectly affects the dynamics via the control input, which is computed using the state estimate $\hat{x}_k$. These discrete time dynamics, therefore, show how the operation mode of the estimator (δ, ϵ) affects the dynamics of the system.

V. Robust Model Predictive Control Solution

In this section, we give an overview of the robust adaptive model predictive controller (RAMPC) that we use in the contract-driven setup of Fig. 3. Here, we consider the estimation errors to be bounded, and use these worst case bounds in the controller formulation. The mathematical details and derivations are available in the online technical report [26]. Experiments confirm that the following controller can be run in real-time, and its computation uses a negligible amount of time relative to the estimation delay.

A. Solution Overview

Recall the operation of the contract-driven control and estimation framework as presented in Section III and Fig. 3. First, the estimator is profiled offline to obtain its delay-error curve, which we denote by Δ. The curve Δ represents a finite number of (δ, ϵ) contracts that the estimator can satisfy. At every time step k, the controller receives a state estimate $\hat{x}_k$ and uses it to compute the control input u_k to be applied to the physical system at time $t_{a,k}$ and the contract $(\delta_{k+1}, \epsilon_{k+1}) \in \Delta$ that will be requested from the estimator at the next step. At $k+1$, the estimator provides an estimate with error at most ϵ_{k+1} and within delay δ_{k+1}. Finally, recall that $J = \sum_{k=0}^{M} (\ell(x_k, u_k) + \alpha \pi(\delta_k))$ combines tracking error and input power in the ℓ terms, and estimation power consumption in the π terms. The scalar α is a design parameter that quantifies the importance of power consumption to the overall performance of the system and is set empirically by the user.

The contract-driven controller's task is to find a sequence of inputs $u_k \in U$ and of contracts $(\delta_k, \epsilon_k) \in \Delta$ such that the cost J is minimized, and the state x_k is always in the set X. The challenge in finding the control inputs is that the controller does not have access to the real state x_k, but only to an estimate $\hat{x}_k$. The norm of the error $e_k = \hat{x}_k - x_k$ is bounded by the contractual ϵ_k, which varies at each time step.

Let us fix the *prediction horizon* $N \geq 1$. Assume that the current contract (under which the current estimate $\hat{x}_k$ was obtained) is (δ_k, ϵ_k), and that the previously applied input is u_{k-1}. To compute the new input value u_k and next contract $(\delta_{k+1}, \epsilon_{k+1})$, the proposed RAMPC seeks to solve the following optimization problem which we denote by $\mathbb{P}_\Delta(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$:

$$\begin{aligned} J^*[0:N] = \min_{\mathbf{u},\mathbf{x},\underline{\delta},\underline{\epsilon}} & \sum_{j=0}^{N} \left(\ell\left(x_{k+j}, u_{k+j}\right) + \alpha\pi\ (\delta_k)\right) \\ \text{s.t. } & \forall j \in \{0, \ldots, N\} \\ & x_{k+j+1} = Ax_{k+j} + B_1(\delta_k)\, u_{k+j-1} + B_2(\delta_k)\, u_{k+j} \\ & \left[x_{k+j+1},\, u_{k+j}\right]' \in X \times U. \end{aligned} \tag{3}$$

Here, RAMPC needs to find the optimal length-N input sequence $\mathbf{u}^* = (u_k^*, \ldots, u_{k+N}^*) \in U^N$, corresponding state sequence $\mathbf{x} = (x_k, \ldots, x_{k+N}) \in X^N$, delay sequence $\underline{\delta} = (\delta_k, \ldots, \delta_{k+N})$ and error sequence $\underline{\epsilon} = (\epsilon_k, \ldots, \epsilon_{k+N})$ such that $(\delta_k, \epsilon_k) \in \Delta$, which minimize the N-step cost $J[0:N]$. The matrices that make up the system dynamics are defined in Section IV-D. As in regular MPC [27], once a solution $\mathbf{u}^*$ is found, only the *first* input value u_k^* is applied to the physical system, thus yielding the next state x_{k+1} as per (2). At the next time step $k+1$, RAMPC sets up the new optimization $\mathbb{P}_\Delta(\hat{x}_{k+1}, \delta_{k+1}, \epsilon_{k+1}, u_{k+1-1})$ and solves it again.

To make this problem tractable, we first assume that the mode is fixed throughout the N-step horizon, that is, $(\delta_{k+j}, \epsilon_{k+j}) = (\delta, \epsilon)$ for all $1 \le j \le N$. Thus, for every value (δ, ϵ) in Δ, we can setup a different problem (3) and solve it. Let $J^*_{(\delta,\epsilon)}$ be the corresponding optimum. The solution with the smallest objective function value yields the input value u_k^* to be applied and the next contract (δ^*, ϵ^*).

Because RAMPC only has access to the state estimate, we extend the RMPC approach in [28] and [29]. Namely, the problem is solved for the *nominal dynamics* which assumes zero process and observation noise ($w_{k+j} = 0$) and zero estimation error ($\hat{x}_{k+j} = x_{k+j}$) over the prediction horizon. Let $\overline{x}$ be the state of the system under nominal conditions. To compensate for the use of nominal dynamics, RMPC replaces the constraint $(\overline{x}_{k+j}, u_{k-1+j}) \in X \times U := \mathcal{Z}$ by $(\overline{x}_{k+j}, u_{k+j}) \in \mathcal{Z}_j(\epsilon_k, \epsilon)$, where $\mathcal{Z}_j(\epsilon_k, \epsilon) \subset \mathcal{Z}$ is $\mathcal{Z}$ "shrunk" by an amount corresponding to ϵ, as explained in the technical report [26]. Intuitively, by forcing $(\overline{x}_{k+j}, u_{k-1+j})$ to lie in the reduced set $\mathcal{Z}_j(\epsilon_k, \epsilon)$, the bounded estimation error and process noise are guaranteed not to cause the true state and input to exit the constraint sets X and U. The tractable optimization for a given (δ, ϵ), denoted by $\mathbb{P}_{(\delta,\epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$, is then

$$\begin{aligned} J^*_{(\delta,\epsilon)} = \min_{\mathbf{u},\mathbf{x}} & \sum_{j=0}^{N} \left(\ell\left(\overline{x}_{k+j}, u_{k+j}\right) + \alpha\pi\ (\delta)\right) \\ \text{s.t. } & \forall j \in \{0, \ldots, N\} \\ & \overline{x}_{k+j+1} = A\overline{x}_{k+j} + B_1(\delta)\, u_{k+j-1} + B_2(\delta)\, u_{k+j} \\ & \left(\overline{x}_{k+j}, u_{k+j}\right) \in \mathcal{Z}_j(\epsilon_k, \epsilon). \end{aligned} \tag{4}$$

Algorithm 1 summarizes the RAMPC algorithm.

We state the following result (proof in technical report [26]).

Theorem 5.1: If at the initial time step there exists a contract value $(\delta, \epsilon) \in \Delta$, an initial state estimate $\hat{x}_0 \in X$, and an input value $u_{-1} \in U$, such that $\mathbb{P}_{(\delta,\epsilon)}(\hat{x}_0, \delta_0, \epsilon_0, u_{0-1})$ is feasible then the system (2) controlled by Algorithm 1 and subjected to disturbances constrained by $w_k \in \mathcal{W}$ robustly satisfies the state constraint $x \in X$ and the control input constraint $u \in U$, and all subsequent iterations of the algorithm are feasible.

VI. Stochastic Model Predictive Control Solution

The control algorithm developed in Section V assumes that the state-estimation error e lies in a bounded set, E. In practice, this can result in a very conservative approximation. Assuming instead that the error arises from a random distribution allows us to develop a *chance constrained* formulation for the controller, outlined in this section. We call this control algorithm the stochastic adaptive model predictive controller (SAMPC). Here, the constraints on the state have to be satisfied with some probability $1 - \zeta$, rather than in a deterministic manner as in the RAMPC formulation.

A. Solution Overview

Starting from the contract-driven control and estimation framework of Section III, we denote the profiled delay-error curve of the estimator by Δ. This curve Δ consists of a finite number of contract options (δ, Σ) that the estimator can satisfy at run-time. Here, $\Sigma \in \mathbb{R}^{n \times n}$ is the positive semi-definite covariance matrix associated with the now stochastic state-estimation error e. It can be obtained through profiling the performance of the estimator as outlined in Section VII. We assume that the mean of the estimation errors is zero in all the contracts, but the formulation and analysis that follows also extends to distributions with nonzero means. δ is again the computation time the estimator takes in a particular mode of operation.

The SAMPC works in a manner similar to the RAMPC. At each time step k, the controller receives a state estimate $\hat{x}_k$ and uses it to compute: 1) the control signal u_k and 2) the contract $(\delta_{k+1}, \Sigma_{k+1}) \in \Delta$ that will be met by the estimator at the following time step. Following this, at time step $k+1$ the estimator gives a state estimate $\hat{x}_{k+1}$ with error $e_{k+1} = \hat{x}_{k+1} - x_{k+1}$ drawn from a distribution with covariance Σ_{k+1} and within time δ_{k+1}.

The cost function to be minimized is $J = \sum_{k=0}^{M}(l(x_k, u_k) + \alpha\pi(\delta_k))$ that combines the tracking error and input power through the l term and the estimator power consumption through the π terms. The SAMPC control algorithm then finds a sequence of control signals u_k and the contracts at each time step $(\delta_k, \Sigma_k) \in \Delta$ such that J is minimized and the state x_k and input u_k respect chance constraints of the form

$$P\left([x_k, u_k] \in X \times U\right) \ge 1 - \zeta. \quad \forall k. \tag{5}$$

Here, $0 < \zeta \le 1$ is a design parameter that decides the lower bound on the constraint satisfaction probability. To achieve these objectives, the SAMPC aims to solve the following optimization (with horizon $N \ge 1$), denoted by $\widetilde{\mathbb{P}}_\Delta(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$, at each time step k:

$$\begin{aligned} J^*[0:N] = \min_{\mathbf{u},\mathbf{x},\underline{\delta},\underline{\Sigma}} & \sum_{j=0}^{N} \left(\ell\left(x_{k+j}, u_{k+j}\right) + \alpha\pi\ (\delta_k)\right) \\ \text{s.t. } & \forall j \in \{0, \ldots, N\} \\ & x_{k+j+1} = Ax_{k+j} + B_1(\delta_k)\, u_{k+j-1} + B_2(\delta_k)\, u_{k+j} \\ & P\left([x_k, u_k] \in X \times U\right) \ge 1 - \zeta. \end{aligned} \tag{6}$$

Similar to the RAMPC, the SAMPC needs to find the optimal length-N input sequence $\mathbf{u} = (u_k^*, \ldots, u_{k+N}^*)$, the corresponding state sequence $\mathbf{x} = (x_{k+1}, \ldots, x_{k+N+1})$, the delay sequence $\underline{\delta} = (\delta_k, \ldots, \delta_{k+N})$ and associated error covariance sequence $\underline{\Sigma} = (\Sigma_k, \ldots, \Sigma_{k+1})$ (such that $(\delta_k, \Sigma_k) \in \Delta$) which

Algorithm 1 RAMPC Algorithm With Anytime Estimation

```
1: $(\delta_0, \epsilon_0)$ and $u_{-1}$ specified by designer
2: Apply $u_{-1}$
3: for $k = 0, 1, \ldots, M$ do
4:     Estimate $\hat{x}_k$ with guarantee $(\delta_k, \epsilon_k)$
5:     for each $(\delta, \epsilon) \in \Delta$ do
6:         $(u_k^*, J_{(\delta,\epsilon)}^*) \leftarrow$ Solve $\mathbb{P}_{(\delta,\epsilon)}(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$
7:     end for
8:     $(\delta^*, \epsilon^*, u_k^*) \leftarrow \text{argmin}_{(\delta,\epsilon)} J_{(\delta,\epsilon)}^*$
9:     Apply control input $u_k = u_k^*$ and estimation mode $(\delta_{k+1}, \epsilon_{k+1}) = (\delta^*, \epsilon^*)$
10: end for
```

minimize the N-step cost $J[0 : N]$ and ensuring the chance constraint of (5) is satisfied.

Consistent with regular MPC framework, once a solution $\mathbf{u}$ is found, only the first input u_k is applied to the system, resulting in state x_{k+1}. At the next time step, after receiving the state estimate $\hat{x}_{k+1}$ from the estimator based on the contract of step k, the SAMPC sets up the new optimization $\widetilde{\mathbb{P}}_\Delta(\hat{x}_{k+1}, \delta_{k+1}, \Sigma_{k+1}, u_{k+1-1})$ and solves it, repeating the process at each subsequent time step.

Similar to RAMPC, the SAMPC only has access to the state estimate, we extend the stochastic MPC (SMPC) approach in [30]. Namely, the problem is solved for the *nominal dynamics* which assume zero process and observation noise ($w_{k+j} = 0$) and zero estimation error ($\hat{x}_{k+j} = x_{k+j}$) over the prediction horizon. Let $\overline{x}$ be the state of the system under nominal conditions. To compensate for the use of nominal dynamics, SMPC replaces the constraint of (5) by $(\overline{x}_{k+j}, u_{k+j}) \in \widetilde{\mathcal{Z}}_j(\Sigma_k, \Sigma)$, where $\widetilde{\mathcal{Z}}_j(\Sigma_k, \Sigma) \subset \mathcal{Z}$ is $\mathcal{Z} = X \times U$ "shrunk" by an amount corresponding to Σ, as explained in the technical report [26]. Intuitively, by forcing $(\overline{x}_{k+j}, u_{k-1+j})$ to lie in the reduced set $\widetilde{\mathcal{Z}}_j(\Sigma_k, \Sigma)$, the stochastic estimation error and process noise are guaranteed to be such that the state and the input respect the joint chance constraint of (5).

The tractable optimization for a given (δ, Σ), denoted by $\widetilde{\mathbb{P}}_{(\delta,\Sigma)}(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$, is then

$$
\begin{aligned}
J_{(\delta,\Sigma)}^* = \min_{\mathbf{u},\mathbf{x}} & \sum_{j=0}^{N} \left(\ell\left(\overline{x}_{k+j}, u_{k+j}\right) + \alpha\pi\left(\delta_k\right)\right) \\
\text{s.t. } & \forall j \in \{0, \ldots, N\} \\
& \overline{x}_{k+j+1} = A\overline{x}_{k+j} + B_1\left(\delta_k\right) u_{k+j-1} + B_2\left(\delta_k\right) u_{k+j} \\
& \left(\overline{x}_{k+j}, u_{k+j}\right) \in \widetilde{\mathcal{Z}}_j\left(\Sigma_k, \Sigma\right).
\end{aligned} \tag{7}
$$

Construction of the shrunk constraint sets $\widetilde{\mathcal{Z}}_j$ is covered in the technical report [26]. In practice, we solve the optimization for each $(\delta, \Sigma) \in \Delta$ in parallel and pick the optimal contract and the corresponding control signal as outlined in Algorithm 1 (solving $J_{(\delta,\Sigma)}^*$ instead of $J_{(\delta,\epsilon)}^*$ in this case). Theorem 6.1 (proven in [26]) states the guarantees of this control algorithm.

Theorem 6.1: For any estimation mode (δ, Σ), if $\widetilde{\mathbb{P}}_{(\delta,\Sigma)}(\hat{x}_k, \delta_k, \Sigma_k, u_{k-1})$ is feasible then the system (2) controlled by the SAMPC and subjected to disturbances constrained by $w_k \in \mathcal{W}$ satisfies, with probability at least $1 - \zeta$, the state constraint $x_k \in X$ and control input constraint $u_k \in U$, and the subsequent optimization $\widetilde{\mathbb{P}}_{(\delta,\Sigma)}(\hat{x}_{k+1}, \delta_{k+1}, \Sigma_{k+1}, u_k)$ are feasible with probability 1.

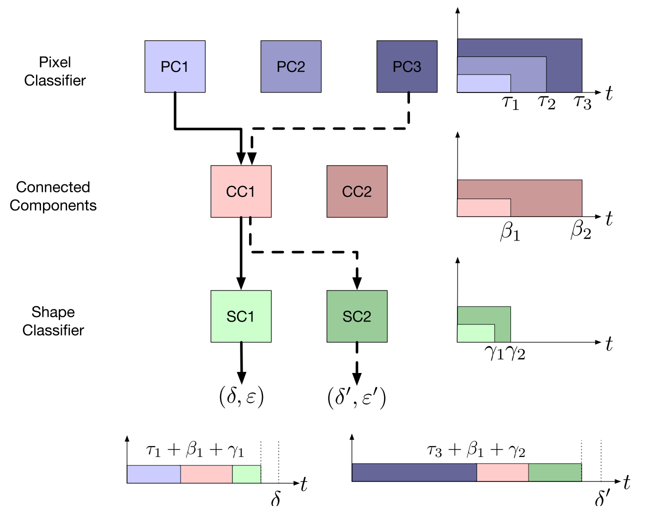


Fig. 5. Illustration of the building blocks used to compose the contract object detector and their representation as real-time tasks. For a given (δ, ϵ) contract, knob settings are chosen at run-time resulting in a schedule to execute these sequential components, or tasks, to respect the contract.

Online, the SAMPC is executed in a manner similar to the RAMPC as outlined in Algorithm 1. Note that having to solve the optimization for each mode (Algorithm 1, line 5) does not add a significant computational burden as these can be solved in parallel. Section VIII shows these methods executing in real-time at a high rate (20 Hz).

VII. Contract-Based Perception Algorithms

We presupposed, in Section III, the existence of an estimation error versus computation delay curve Δ for the state estimator. The controller uses this curve at each discrete time step to select the operating mode (δ, ϵ) for the estimator at the next time step, as seen in Section V. In this section, we show how this curve can be obtained for perception-based algorithms through a general approach. We demonstrate this on two applications (object detection and visual odometry), and we show ways for the contract-based estimation algorithm to realize the points on the curve at runtime.

A. Profiling and Creating an Anytime Contract Based Perception-and-Estimation Algorithm

In order to profile a contract estimator, we first need to identify the distinct building blocks (or tasks) of the perception algorithm. We then find the relevant parameters, or *knobs*, in each task (e.g., maximum iterations in a loop) such that varying them results in changes in the computation time (δ) and the quality (ϵ) of the overall output of the estimation algorithm.

This procedure is tested through implementation on a CV-based object detection toolchain, an overview of which is shown in Fig. 5. This object recognition toolchain is tasked with tracking an object of interest (OOI) across the frames of a video stream. The first level of this is a pixel classifier that assigns a probability for each pixel being a part of the OOI. Applying this to a given frame and thresholding the probabilities for each pixel over some minimum probability results in a binary image with the pixels of interest taking a value 1, others being 0. The second level involves denoising the binary image, and then finding the connected components (CCs), that is, collecting adjacent pixels of interest into (possibly disconnected) objects. The third and final level is a shape classifier that is run on the output of the CCs to determine whether each object from it is of interest or not.

Our implementation uses a Gaussian mixture model (GMM) for the pixel classifier and the shape classifier. The *knob* here is the number of Gaussian distributions in the GMM for the pixel classifier and the number of features for the shape classifier. A smaller number of Gaussians or features (i.e., the dimension of the Gaussian) will result in a faster, but possibly inaccurate classifier. On the other hand, more Gaussians, or features, can result in improved performance, but at the cost of higher computation time. As is typically done, knob values that result in an overfit are identified and rejected via cross-validation during the training process.

The filtering for denoising the binary image, and the CCs algorithm form the second level of the object recognition toolchain and the knob here consists of selecting either a four-connected or eight-connected implementation.

In this implementation, the number of knob settings for the object recognition tool chain is $K = $ (#Gaussians for pixel classifier $\times$ #neighbors for CC $\times$ #features for shape classifier), and has a total of $3\times2\times2 = 12$ values.

The tradeoff curve for the entire toolchain is obtained by profiling all 12 knob settings by running it on a data set for profiling. Through this process we obtain, for each of the different knob values: 1) the output quality error ϵ and 2) the computation times δ for the entire toolchain. This offline gathering of information gives us the information to be used at run-time in the codesign framework. The profiled performance of the CV-based object recognition toolchain considered here is shown in Fig. 6.

It should be noted that for each block of the toolchain, the relation between knob value and runtime/quality of output is not necessarily monotonic. Fig. 6 shows the mean perception error[2] and the 90th percentile execution time for the different knob settings. Although the trend is that perception error decreases with increasing execution time, there are some knob settings leading to both larger perception error and larger execution time, which is seen in the nonmonotonic behavior seen in Fig. 6.

B. Run-Time Execution of the Contract-Driven Perception Algorithm

After profiling the contract-driven estimator, we can use the information at run-time to choose which knob settings are needed to respect a given (δ, ϵ) contract. This is tantamount to choosing altered versions of tasks and scheduling them to execute one after the other in a predefined manner to optimally perform the job of detecting an OOI. Fig. 5 shows the various tasks and their different versions for every knob setting and the resulting task schedules.

C. Visual Odometry

An example of a vision-based state estimation algorithm is Semi-Direct Monocular Visual Odometry (SVO) [1], which we will use in Section VIII to get state estimates for control of the hexrotor robot. SVO detects *corners* in an image and

[2]An error is a distance between the true centroid and the estimated centroid of the OOI.

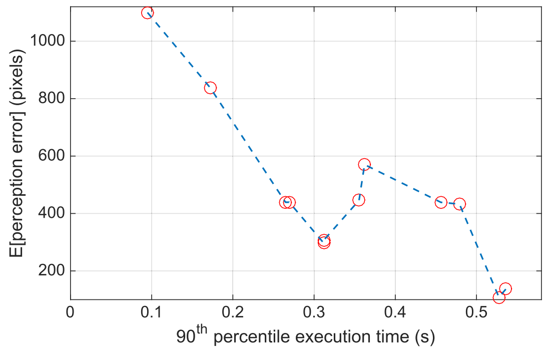


Fig. 6. Profiled delay-error curve for the object detection tool chain run at different parameter settings.

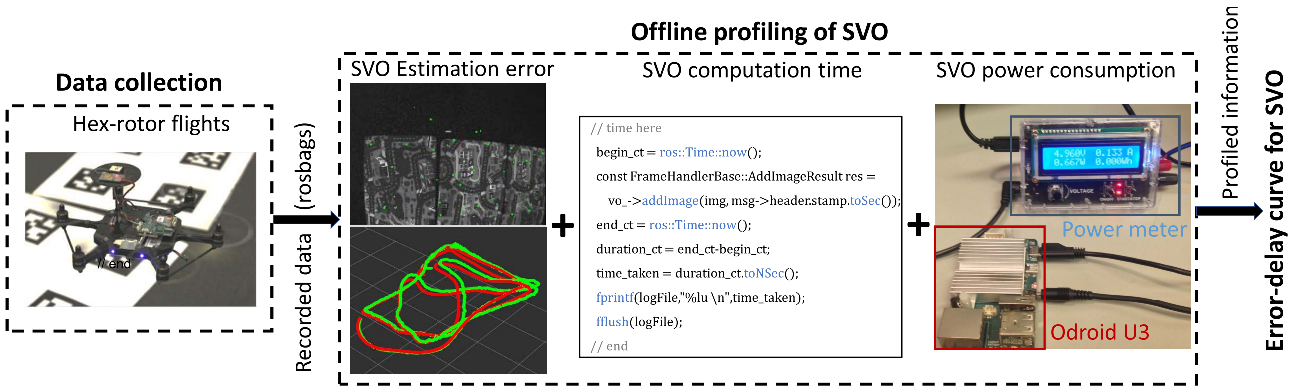
tracks them across consecutive frames of a video feed in order to localize the moving robot and generate a state estimate. Since this state estimate is used for closed-loop control of the hexrotor, SVO has to run in real-time at a frame rate that is fast enough for the purpose of controlling a flying robot. The number of corners $\#C$ (as well as their quality) being tracked from frame to frame affects the computation time of the localization algorithm and the resulting quality of the state estimate. In general, assuming that the camera is looking at a feature-rich environment, detecting and tracking a higher number of corners result in better localization accuracy but also take larger computation time. For the profiling of SVO, the number of corners $\#C$ is the only knob and is varied to obtain an error-delay curve of the localization performance.

1) Profiling SVO Performance: Fig. 7 outlines the profiling process for SVO. We start with the hexrotor, running robot operating system (ROS), flying (either manually or autonomously) in an environment with a *Vicon* motion-capture system [31]. Throughout the flight, the downward-facing monocular camera captures frames at the desired rate of 20 Hz. We also log the IMU data, as well as the high-accuracy six-DOF pose estimate generated by the motion capture system, which we will use as the ground truth for the hexrotor positions and velocities. We collected data, recorded as rosbags, over 15 min of flights with randomly chosen paths. These flights for data collection can be performed either autonomously or with manual control, for example, in case an autonomous controller has not been formulated. In our case, we collect data through a combination of both.

After the data collection, which is a one-time process, we process the data offline with SVO running at the desired settings of $\#C$ and compare state estimates from the SVO to the high-accuracy ones from the vicon to get the state estimation error profile. We also measure the SVO computation time for each setting of $\#C$, as well as the power consumed by the Odroid-U3 computation platform [32], a stripped-down version of which is used on the hexrotor. More details on the process are in the online technical report [26].

Through this offline profiling process, we avoid having to fly separate flights to get profiling information for every knob setting ($\#C$), and the result of this profiling is used in the formulation of the controller and used at run-time by it to generate contracts for the contract-driven estimator (Fig. 3).

2) Error-Delay Curve for SVO: Obtained from the profiling process outlined above, Fig. 8 shows the error-delay curve(s) of the localization error (in positions) of the hexrotor with



```
// time here
begin_ct = ros::Time::now();
const FrameHandlerBase::AddImageResult res =
    vo_->addImage(img, msg->header.stamp.toSec());
end_ct = ros::Time::now();
duration_ct = end_ct-begin_ct;
time_taken = duration_ct.toNSec();
fprintf(logFile,"%lu \n",time_taken);
fflush(logFile);
// end
```

Fig. 7. Profiling process to characterize the performance of SVO in terms of estimation error, computation time, and power consumption. Sensor and ground-truth data are logged from flights of the hexrotor and then played back and processed offline to generate the error-delay curve (shown in Fig. 8) for SVO. The code snippet shows how little modification is needed to the SVO code base to be able to profile its timing characteristics. Through this offline profiling process, we avoid the need of performing separate flights for each knob setting of SVO.

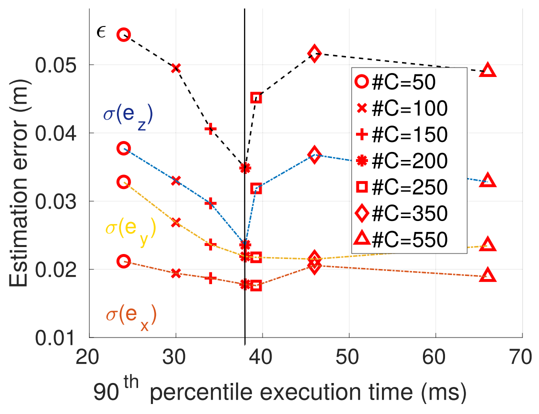


Fig. 8. (Color online) Error-delay curve for the SVO algorithm running on the Odroid-U3 with different settings of maximum number of features ($\#C$) to detect and track. The vertical line shows the cut-off for maximum delay and the SVO settings that are allowable (up to $\#C = 200$) for closed-loop control of a hexrotor at 20 Hz. No value of $\#C$ is used above this as it results in the delay approaching the sampling period of the controller.

SVO running on an Odroid-U3. The curve, obtained through data collected over multiple flights in a fixed environment, shows the worst case error ϵ (over all flights and all components of the 3-D position, used in Section V), as well the standard deviation of the error for all components of the 3-D position (used in the stochastic control formulation of Section VI) versus the computation time δ for varying number of corners being tracked $\#C$. δ is obtained by considering the 90th percentile of computation times, while ϵ is obtained by computing the infinite norm of the 90th percentile error over the three components (x, y, z) of the position. Note that as the number of corners being tracked increases, the computation time increases and the estimation error decreases as expected, but only up to a point. At $\#C = 250$, the estimation error increases. We hypothesize that this is due to the decreasing quality of the corners in the environment now being tracked. This is because if the scene is not particularly feature-rich, and a sizable fraction of the $\#C$ corners are of poor quality (i.e., unstable or hard to track across frames), and we can expect the localization error to increase as the poor quality of the corners detected adds noise to the visual odometry estimates.

The profiling approach outlined here is generally applicable to most perception-based estimators. One limitation, however, is that the framework applies only to the environment that the profiling has been carried out in. Additionally in some applications, data collection might be too costly.

VIII. Case Study: Feedback Control of a Hexrotor Robot

To evaluate the performance of our proposed methods, we implemented the contract-driven estimator and control scheme on a hexrotor robot (Fig. 7). It is equipped with a downward-facing camera, allowing us to use SVO for localization. The onboard computation platform is an Odroid U-3 [32] computer running Ubuntu as the operating system. The computer also runs an ROS [33] which is responsible for executing the estimation and control algorithm at a fixed rate, and the communication between them. More details can be found in the online technical report [26].

A. Experiment Design

To compare the performance of the RAMPC and SAMPC algorithms developed in this article with that of an MPC that does not leverage codesign, we task the controllers with the following two predefined reference trajectories, shown in Fig. 9. The reference trajectories are generated using the jerk minimizing trajectory generator of [34].

1) *The Hourglass Trajectory:* This trajectory involves flying straight lines between the desired waypoints, as shown in Fig. 9. In order to get the straight lines, the waypoints are associated with desired velocities of zero (in each axis). The duration of this trajectory is around 14 s. The entire trajectory is flown at a constant height of 1 m. A video of the hexrotor flying this trajectory can be found at https://youtu.be/-ltJO2gVxWs.
2) *Spiral in x, y With Sinusoidal Variations in z:* This trajectory consists of smooth curves between waypoints, with the waypoints such that in the xy-plane the trajectory looks like a spiral converging toward the origin, while in the z-axis it consists of sinusoidal variations along a reference height of 1 m. The duration of this trajectory is 17 s. A video of the hexrotor flying this trajectory is at https://youtu.be/hmTRxrq4NJg.

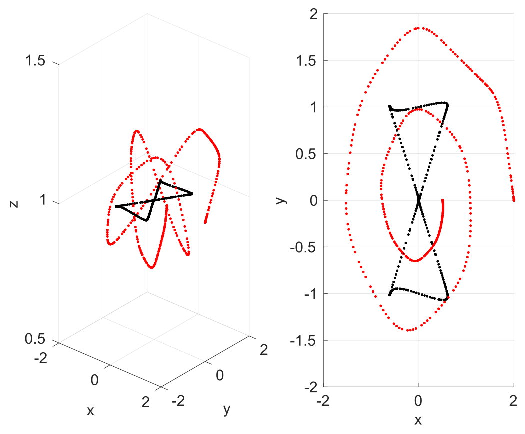


Fig. 9. (Color online) Two reference trajectories, the spiral is in red dashed line and the hourglass is in black solid line. Right: trajectories projected on the xy plane. Note, the spiral starts on the outside and ends inward while the hourglass trajectory starts and ends at (0, 0, 1).

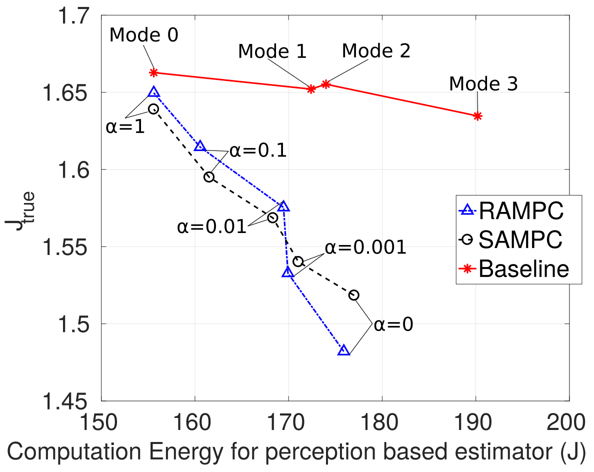


Fig. 10. Performance, hourglass trajectory. The vertical axis has the average control performance (8) over the flights for the labeled settings, with lower values implying better control performance. The horizontal axis shows the computation power (in Joules) consumed by SVO to perform the state estimation task. The figure shows how our methods (RAMPC/SAMPC) leveraging the codesign have both better control performance while consuming less computation power than the baseline method.

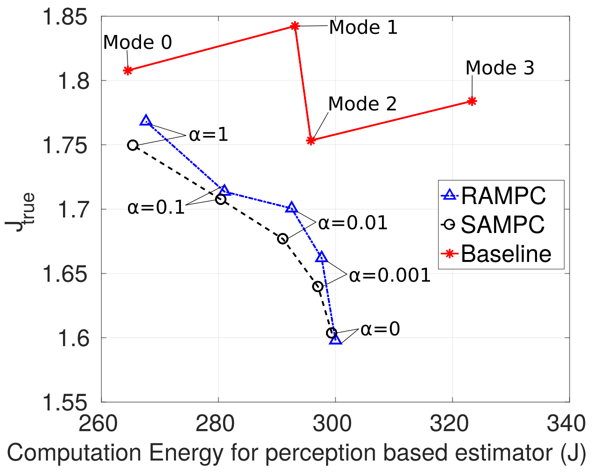


Fig. 11. Performance, spiral trajectory. The vertical axis has the average control performance (8) over the flights for the labeled settings, with lower values implying better control performance. The horizontal axis shows the computation power (in joules) consumed by SVO to perform the state estimation task. Similar to the case for the hourglass trajectory, our methods outperform the baseline.

These trajectories are flown with: 1) the **baseline**, a robust MPC formulation that does not leverage the codesign of computation and control, with all four chosen modes of SVO used for the state feedback; 2) the **RAMPC** algorithm with varying values of α, the weight for the computation power in the optimization; and 3) the **SAMPC** (with $\zeta = 0.82$) with varying values of α. Each trajectory is flown twice for each one of these settings to get a comparison of control performance and computation energy consumption. This leads to a total of 56 flights to gather the data presented in this case study.

B. Experimental Results

To measure the performance of the controllers in a standardized manner, we used the following measure of control performance:

$$J_{\text{true}} = \frac{1}{T_{\max}} \sum_{k=0}^{T_{\max}/h} \left(x_k - x_k^{\text{ref}}\right)^T Q \left(x_k - x_k^{\text{ref}}\right) + u_k^T R u_k. \quad (8)$$

Here, x^{ref} is the desired trajectory, and Q and R are the matrices used in the cost of MPC/RAMPC/SAMPC, h is the sampling time (50 ms) and $T_{\max}$ is the duration of the particular trajectory flown. J_{true} can be accurately evaluated as we have access to the true state, x_k, from the Vicon system.

1) Comparison to the Baseline: Fig. 10 shows the control performance and the SVO energy consumption for the hourglass trajectory for the baseline RMPC, RAMPC, and SAMPC for different settings. The SAMPC and RAMPC result in lower (average across flights) values of J_{true} than the baseline controller, that is, better control performance. As the value of α increases, the power consumption decreases and the control performance degrades for the RAMPC and SAMPC. This is expected as α is the weight for the computation power in the overall optimization cost of (3) [and (6)] and increasing it would make computation power more important relative to the control performance. Fig. 11 shows similar behavior for the spiral trajectory. The notable exception is in the baseline performance, where the most accurate mode (mode 3) of SVO does not result in the best control performance of the fixed mode RMPC controller. This is possibly because the spiral trajectory is more aggressive than the hourglass trajectory, which involves stopping at each corner waypoint of the trajectory, and spending time in mode 3 comes with a computation delay that degrades the control performance despite the increased accuracy of the state estimate. For either trajectory, SAMPC and RAMPC give a better control performance than the baseline for the corresponding computation energy consumption. For both cases, the control performance of SAMPC and RAMPC are close to each other, with the SAMPC performing slightly better for the spiral trajectory.

Summary: Across both the trajectories, the best case control performance of our methods results in about a 10% improvement compared to that of the baseline. To achieve this performance, our methods result in SVO using about 5%–6% less computation energy compared to the baseline (at the setting resulting in best control performance).

TABLE I
SVO MODES USED IN THE EXPERIMENTS

Mode	$\#C$	$\delta\,[ms]$	$\epsilon\,[m]$	$\sigma(e_x)$	$\sigma(e_y)$	$\sigma(e_z)$	$\pi(\delta)[mW]$
0	50	24	0.054	0.021	0.033	0.038	778
1	100	30	0.049	0.019	0.027	0.033	862
2	150	34	0.041	0.019	0.024	0.030	870
3	200	38	0.035	0.018	0.022	0.024	951

TABLE II
FRACTION OF TIME SPENT IN MODES: HOURGLASS TRAJECTORY, RAMPC

	Mode 0	Mode 1	Mode 2	Mode 3
$\alpha = 0$	0.398	0.008	0.024	0.570
$\alpha = 0.001$	0.523	0.004	0.024	0.440
$\alpha = 0.01$	0.557	0.000	0.067	0.374
$\alpha = 0.1$	0.820	0.000	0.055	0.123
$\alpha = 1$	1.000	0.000	0.000	0.000

TABLE III
FRACTION OF TIME SPENT IN MODES: HOURGLASS TRAJECTORY, SAMPC

	Mode 0	Mode 1	Mode 2	Mode 3
$\alpha = 0$	0.374	0.000	0.004	0.621
$\alpha = 0.001$	0.514	0.016	0.051	0.418
$\alpha = 0.01$	0.617	0.000	0.032	0.351
$\alpha = 0.1$	0.793	0.000	0.076	0.131
$\alpha = 1$	1.000	0.000	0.000	0.000

TABLE IV
FRACTION OF TIME SPENT IN MODES: SPIRAL TRAJECTORY, RAMPC

	Mode 0	Mode 1	Mode 2	Mode 3
$\alpha = 0$	0.381	0.015	0.015	0.589
$\alpha = 0.001$	0.422	0.012	0.018	0.548
$\alpha = 0.01$	0.504	0.000	0.041	0.455
$\alpha = 0.1$	0.680	0.000	0.082	0.238
$\alpha = 1$	0.995	0.000	0.015	0.000

TABLE V
FRACTION OF TIME SPENT IN MODES: SPIRAL TRAJECTORY, SAMPC

	Mode 0	Mode 1	Mode 2	Mode 3
$\alpha = 0$	0.396	0.003	0.018	0.584
$\alpha = 0.001$	0.434	0.009	0.018	0.540
$\alpha = 0.01$	0.531	0.000	0.038	0.431
$\alpha = 0.1$	0.695	0.000	0.073	0.232
$\alpha = 1$	0.971	0.000	0.029	0.000

This clearly demonstrates the benefit of the codesign between the perception-based estimation and the control algorithms.

2) Impact of the Weight for Computation Power (α): As α takes on a high value, the control performance of RAMPC and SAMPC for the hourglass trajectory approaches that of the baseline RAMPC with SVO mode fixed to 0. This is backed up in the observation of Tables II and III which show that for $\alpha = 1$, the RAMPC and SAMPC select mode 0, the low-power but high estimation error mode, of SVO all the time. Tables II and III show the fraction of time spent in each mode of SVO as α changes. Note that as α, the weight for the computation power, increases the time spent in the low power mode 0 also increases while the time spent in the more accurate but higher power modes accordingly decreases. Similar behavior is noted for the spiral trajectory, and Tables IV and V show the fraction of time spent in the different SVO modes as α changes for RAMPC and SAMPC flying the spiral trajectory respectively.

3) Snapshots of the Control Performance of RAMPC and SAMPC: Fig. 12 shows the reference and actual positions of the hexrotor (in x-, y-, and z-coordinates) as functions of time

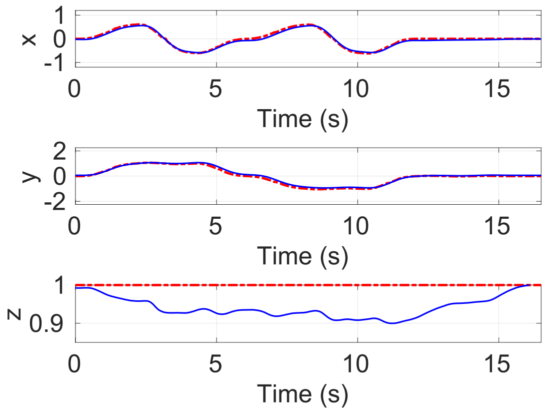


Fig. 12. (Color online) Reference positions (red dashed line) and actual positions (blue line) of the hexrotor flying the hourglass trajectory while being controlled by the SAMPC ($\alpha = 0$).

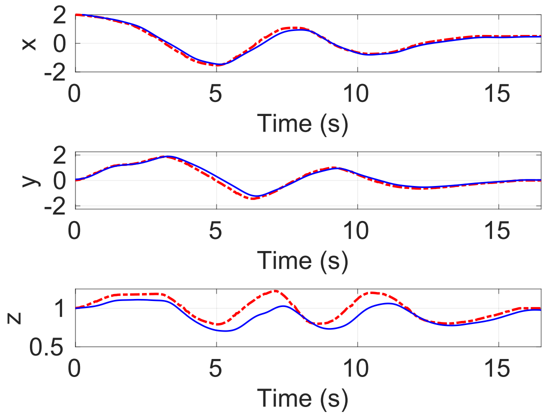


Fig. 13. (Color online) Reference positions (red dashed line) and actual positions (blue line) of the hexrotor flying the spiral trajectory while being controlled by the RAMPC ($\alpha = 0.1$).

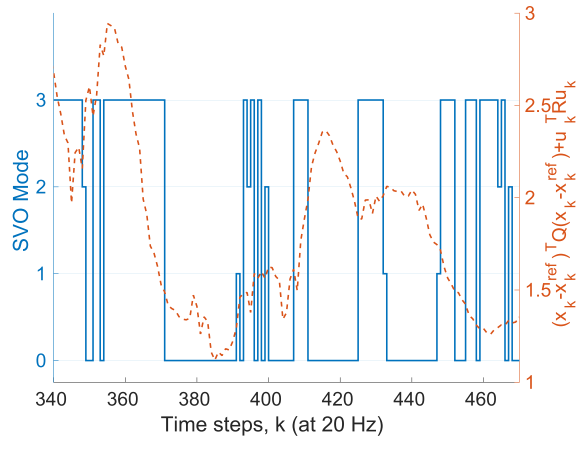


Fig. 14. SVO Mode and control cost (per time-step) over time for the spiral trajectory flown with SAMPC at $\alpha = 0.001$.

for the hourglass trajectory controlled by the SAMPC ($\alpha = 0$). Note the near-perfect tracking in x and y. The small dip in the height (z-coordinate) is due to a combination of model error (due to inaccuracy of the mass) as well as the effect of linearization around hover. Fig. 13 shows the reference and actual positions versus time for the RAMPC ($\alpha = 0.1$) flying the spiral trajectory, showing similarly good tracking performance as in the hourglass trajectory. Fig. 14 shows a snapshot of the SVO mode and tracking cost over time for the spiral trajectory.

IX. Conclusion

In this article, we presented a contract-driven methodology for codesign of estimation and control for autonomous systems. The basic idea is that the control algorithm requests a delay and estimation error (δ, ϵ) contract that the perception-and-estimation algorithm realizes. The control algorithm we designed aims to set time-varying contracts to maximize a performance function while respecting feasibility constraints and stability under the time varying execution delay and estimation error from the estimator. We also illustrate how the contract-driven perception-and-estimation algorithm is designed offline and used at run-time to best meet the (δ, ϵ) contracts set for it. Through a case study on a flying hexrotor, we showed the applicability of our scheme to a real-time closed-loop system. The experimental results show the good performance of our scheme and how it outperforms regular model predictive control which does not leverage codesign. A key result showed how our closed-loop solution is more energy-efficient than MPC while achieving better tracking performance. A focus of ongoing research is to overcome the necessity of the contracts always being met by the estimator as well as on developing an automated toolchain to profile perception algorithms commonly used in autonomous systems.

Acknowledgment

The authors thank Kuk Jang for his help with the figures.

References

[1] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 15–22.

[2] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[3] R. Wilhelm *et al.*, "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, p. 36, 2008.

[4] M. Boddy and T. Dean, "Solving time-dependent planning problems," in *Proc. 11th Int. Joint Conf. Artif. Intell.*, 1989, pp. 979–984.

[5] Y. V. Pant, H. Abbas, K. Mohta, T. X. Nghiem, J. Devietti, and R. Mangharam, "Co-design of anytime computation and robust control," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2015, pp. 43–52.

[6] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI Mag.*, vol. 17, no. 3, p. 73, 1996.

[7] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artif. Intell.*, vol. 172, no. 14, pp. 1613–1643, Sep. 2008.

[8] M. P. Wellman and C. -L. Liu, "State-space abstraction for anytime evaluation of probabilistic networks," *Uncertainty Proc.*, pp. 567–574, Jul. 1994.

[9] R. Mangharam and A. A. Saba, "Anytime algorithms for GPU architectures," in *Proc. IEEE 32nd Real-Time Syst. Symp.*, Nov. 2011, pp. 47–56.

[10] Y. V. Pant *et al.*, "Power-efficient algorithms for autonomous navigation," in *Proc. Int. Conf. Complex Syst. Eng. (ICCSE)*, Nov. 2015, pp. 1–6.

[11] D. E. Quevedo and V. Gupta, "Sequence-based anytime control," *IEEE Trans. Autom. Control*, vol. 58, no. 2, pp. 377–390, Feb. 2013.

[12] R. Bhattacharya and G. J. Balas, "Anytime control algorithm: Model reduction approach," *J. Guid., Control, Dyn.*, vol. 27, no. 5, pp. 767–776, Sep. 2004.

[13] D. Fontanelli, L. Greco, and A. Bicchi, "Anytime control algorithms for embedded real-time systems," in *Hybrid Systems: Computation and Control*. Berlin, Germany: Springer, 2008, pp. 158–171.

[14] V. Narayanan, M. Phillips, and M. Likhachev, "Anytime safe interval path planning for dynamic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 4708–4715.

[15] D. K. Jha, M. Zhu, Y. Wang, and A. Ray, "Data-driven anytime algorithms for motion planning with safety guarantees," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2016, pp. 5716–5721.

[16] S. Choudhury, "Anytime geometric motion planning on large dense roadmaps," M.S. thesis, Robot. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Jul. 2017.

[17] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1478–1483.

[18] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 1–8.

[19] Y. V. Pant, H. Abbas, and R. Mangharam, "Robust model predictive control for non-linear systems with input and state constraints via feedback linearization," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, Dec. 2016, pp. 5694–5699.

[20] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2014, pp. 53–62.

[21] D. D. Niz, L. Wrage, N. Storer, A. Rowe, and R. Rajkumar, "On resource overbooking in an unmanned aerial vehicle," in *Proc. IEEE/ACM 3rd Int. Conf. Cyber-Phys. Syst.*, Apr. 2012, pp. 97–106.

[22] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for QoS management," in *Proc. IEEE RTSS*, Dec. 1997, pp. 298–307.

[23] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. Accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (SIGSOFT/FSE)*, 2011, pp. 124–134.

[24] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *Proc. ACM SIGPLAN Int. Conf. Object Oriented Program. Syst. Lang. Appl.*, 2013, pp. 33–52.

[25] R. St. Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 505–516.

[26] Y. V. Pant *et al.*, "Technical report: Anytime computation and control for autonomous systems," Dept. Elect. Syst. Eng., Univ. Pennsylvania, Philadelphia, PA, USA, Tech. Rep. UPenn-ESE-04-19, Apr. 2019. [Online]. Available: https://bit.ly/33vdNDj

[27] E. Camacho and C. Bordons, *Model Predictive Control*. New York, NY, USA: Springer-Verlag, 2004.

[28] A. Richards and J. How, "Robust model predictive control with imperfect information," in *Proc. Amer. Control Conf.*, Jun. 2005, pp. 268–273.

[29] L. Chisci, J. A. Rossiter, and G. Zappa, "Systems with persistent disturbances: Predictive control with restricted constraints," *Automatica*, vol. 37, no. 7, pp. 1019–1028, Jul. 2001.

[30] B. Kouvaritakis, M. Cannon, Q. Cheng, and S. V. Raković, "Explicit use of probabilistic distributions in linear predictive control," in *Proc. UKACC Int. Conf. Control*, Sep. 2010, pp. 1–6.

[31] *Motion Capture Systems—Vicon*. Accessed: Oct. 30, 2018. [Online]. Available: https://www.vicon.com

[32] ODROID-U3. Accessed: May 13, 2015. [Online]. Available: http://odroid.com/

[33] M. Quigley *et al.*, "Ros: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, p. 5.

[34] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Trans. Robot.*, vol. 31, no. 6, pp. 1294–1310, Dec. 2015.