Co-Design of Anytime Computation and Robust Control

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

***Abstract*— Control software of autonomous robots has stringent real-time requirements that must be met to achieve the control objectives. One source of variability in the performance of a control system is the execution time and accuracy of the state estimator that provides the controller with state information. This estimator is typically perception-based (e.g., Computer Vision-based) and is computationally expensive. When the computational resources of the hardware platform become overloaded, the estimation delay can compromise control performance and even stability. In this paper, we define a framework for co-designing anytime estimation and control algorithms, in a manner that accounts for implementation issues like delays and inaccuracies. We construct an anytime perception-based estimator from standard off-the-shelf Computer Vision algorithms, and show how to obtain a trade-off curve for its delay vs estimate error behavior. We use this anytime estimator in a controller that can use this trade-off curve at runtime to achieve its control objectives at a reduced energy cost. When the estimation delay is too large for correct operation, we provide an optimal manner in which the controller can use this curve to reduce estimation delay at the cost of higher inaccuracy, all the while guaranteeing basic objectives are met. We illustrate our approach on an autonomous hexrotor and demonstrate its advantage over a system that does not exploit co-design.**

I. Introduction

Real-time control of physical systems, like autonomous robots, raises a number of timing and control-related issues at the interface between the controller that's providing the actuation and the estimator that's providing periodic state estimates to the controller. Some of these issues have to do with the inaccuracies introduced by the software implementation of both controller and estimator on a given hardware platform. Specifically, controllers are typically designed to accomplish the functional goals of the system under simplifying assumptions on the quality of the state estimate (e.g., no or fixed error), the estimation delay (e.g., no or fixed delay), and the actuation jitter (e.g., no jitter). Conversely, estimation algorithms are typically designed without regard to how their estimates will be used and under what operating conditions. In particular, an estimator will often *run to completion*: that is, its stopping criteria are designed to provide the best estimate, regardless of runtime or energy consumption. The problem addressed here is that as the real-time requirements on the closed-loop system become more stringent, this separation in the design and execution of controller and estimator can lead to degraded performance, as will be shown in Example 1. The goal of this paper is to present a rigorous framework for the joint design of the controller and estimator, in which the estimator explicitly presents a range of execution time/estimate error operating modes, and the controller switches between these modes in real-time to maintain control performance and reduce energy consumption.

*This work was supported by STARnet a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

The Departments of Electrical and Systems Engineering and Computer and Information Sciences, University of Pennsylvania, Philadelphia, U.S.A. {yashpant,kmohta,habbas, nghiem,rahulm}@seas.upenn.edu, devietti@cis.upenn.edu

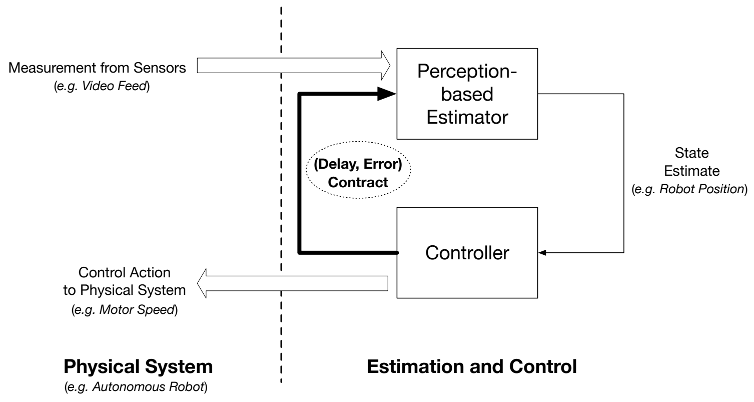

Fig. 1. Contract-based controller and estimator.

Typical design practice determines the Worst-Case Execution Time (WCET) of the estimation task, and engineers the system to satisfy deadlines under WCET conditions. However, the actual execution time of such estimators is heavily dependent on the actual data being processed. So WCET considerations, whether computed online or offline, produce a conservative design. Moreover, classical timing analysis does not guarantee *functional* correctness of the closed-loop system. In addition, the best estimate is not always needed: sometimes a lower quality estimate, obtained with a smaller energy cost, is sufficient to achieve the control objectives. Finally, when obtaining better estimates requires longer runtimes of the estimation task, it may actually be detrimental to ask for the best estimate. For example, when the computational resources are overloaded, there may be a need to spend less time computing a state estimate.

Example 1: To illustrate the impact of estimation delay δ and estimate inaccuracy ϵ on control performance, we show a simple PID controlling the motion of a point mass in the (x, y) plane. The position of the point mass must follow a reference constant trajectory, whose x dimension is shown in Fig. 2 (the same plot can be obtained for the y position). We simulate three cases of estimation (and therefore actuation) delay and error, where a larger delay value δ implies a smaller estimation error ϵ. As can be noted in Fig. 2, the

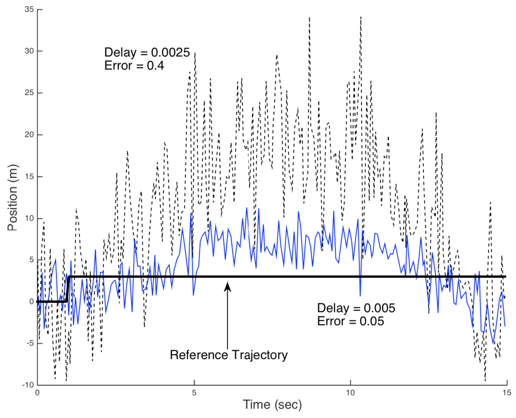

Fig. 2. Effect of delay, error values on control performance.

effect of delay can be non-negligible. Moreover, decreasing delay doesn't necessarily imply better tracking performance: the effect of the concomitant estimation error must be taken into account. In this example, it can be seen that the increasing error 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. 2. Therefore, there is a need to rigorously quantify the trade-off between computation time and estimation error, then exploit that trade-off to achieve the best control performance under the problem constraints. Rather than always run the estimation task to completion, it is useful to have several estimation tasks with varying utilities (i.e., varying delay/error trade-offs). These can then be used at runtime to satisfy the control objectives. □

In this work, we develop the above remarks into a *co-design framework* for a real-time control systems, where the controller and estimator communicate via *contracts*. A contract is a guarantee requested by the controller, and fulfilled by the estimator, that the latter can provide an estimate with a certain maximum error ϵ, and within a certain deadline δ. Both the deadline and the error bound are part of the contract. Using these contracts, we show how the controller can throttle the execution time of the estimation task to preserve good performance and to reduce energy consumption. Our work focuses on estimators that incorporate computationally intensive Computer Vision (CV) algorithms, such as those used in autonomous robot navigation. We refer to these as *perception-based estimators*. Our experiments validate that the execution time of these algorithms is significant and far exceeds the computation time of the control software, and can have an effect on control performance.

Fig. 1 presents the proposed structure of contract-based estimation and control. It shows a traditional feedback loop incorporating estimator, controller and the physical system, augmented with the (Delay, Error) contract between controller and estimator. This contract forms the basis of the proposed approach.

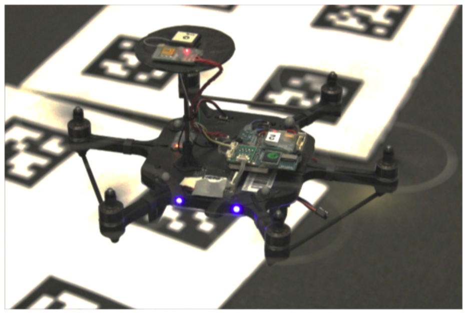

Fig. 3. Autonomous hexrotor with downward-facing camera flying over synthetic features.

Summary of contributions. We present a contract-based framework for the co-design of real-time controller and estimator algorithms, consisting of:

- a well-defined interface between control and estimation, in the form of operating modes or *contracts* on the accuracy and delay provided by the estimator (Section III),
- a controller design that can vary the accuracy and delay of the estimation to achieve control objectives at a lower energy cost (Sections IV, V), and
- a general procedure to compose run-to-completion estimation algorithms into a contract-based estimator (Section VI).
- We illustrate our approach on an autonomous flying robot (shown in Fig. 3) and demonstrate performance and energy gains using our approach over a classical controller (Section VII).

II. Related Work

Anytime algorithms [1] are a class of algorithms that can be interrupted at any point during their execution and still return a usable solution, usually with a monotonically improving quality with time. Contract algorithms [2] are one class of anytime algorithms where the interruption time is pre-determined for any given execution. Our approach, while similar to contract algorithms in the timing aspect, differs significantly as the meaning of a contract expands to including both time *and quality of the solution* (estimation error in our case).

Anytime algorithms have notably been studied for graph search [3], evaluation of belief networks [4] and GPU architectures [5].

As overloaded real-time systems are becoming increasingly common, anytime algorithms for control have become a topic of research interest. Most notably, Quevedo and Gupta [6], Bhattacharya and Balas [7], and Fontanelli et al. [8] have contributed to the topic. Our approach differs significantly from these works as the anytime computation assumption is on the perception-and-estimation algorithm and our controller is a robust controller which can switch between different operating modes of the anytime estimator. Also, while most of these works require either access to

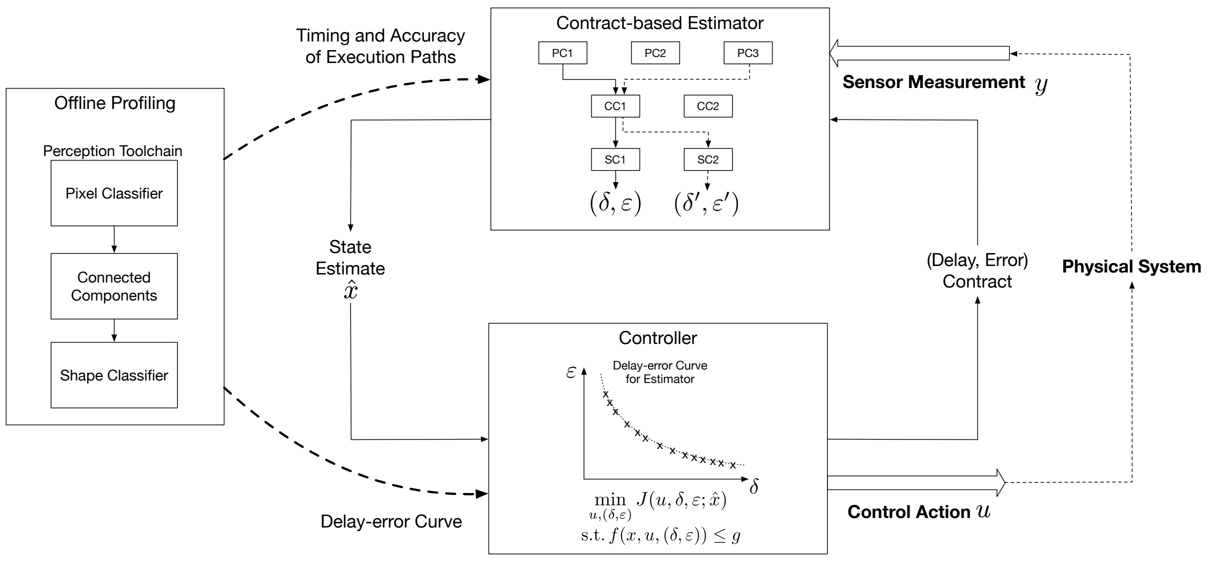

Fig. 4. Contract-based estimator and controller

the full state of the system or have a fast estimator giving them the state estimate [7], our algorithm accounts for the computation time/error of the perception-and-estimation algorithms that are common in autonomous systems.

In real-time systems, recent work [9] uses Typical Worst Case Analysis of the software and Logical Execution Time semantics to provide the controller with knowledge of the timing characteristics of the implementation. Our work, by contrast, profiles the estimation software directly to obtain timing and accuracy information. Whereas [9] is concerned with formal verification of a given controller, we *design* controllers to take advantage of delay/accuracy trade-offs in real-time. The effect of increasing computation time of a task on performance of a UAV has been explored in [10] by using a resource allocation algorithm similar to QRAM [11]. Our work differs from this as we consider the execution time of a task, the estimator, which is directly related to the control performance of a closed loop system and also formulate a control problem around it that provides mathematical guarantees on the performance of the closed loop system.

Also, in the field of computer architecture approximate computing approaches [12], [13], [14] have been studied, seeking time or energy savings by performing a computation approximately instead of precisely. While anytime algorithms and approximate computing share a high-level goal, approximate computing approaches are run-to-completion and also lack a feedback mechanism to permit computation and resources to be balanced dynamically. Additionally the time and energy scale that our approach works at is much higher than what approximate computing looks at.

III. Co-design of estimation and control

In a traditional control system, the controller is unaware of the implementation details of the estimation module and the estimation module is unaware of the requirements of the controller. For example, the design of a feedback controller might not take into account the fact that obtaining a state estimate from a video feed will take a non-negligible amount of time, which we refer to as the estimation delay. Conversely, the design of the perception and estimation might not in general take into account the varying real-time constraints that the controlled system must satisfy. In order to improve performance of real-time closed loop systems using computationally and power limited platforms, we propose the *co-design* of estimation and control. The co-design involves using a *contract-based framework* for both estimator and controller. Namely, the controller requests the estimator to provide a state estimate within a certain deadline δ seconds and with a certain error bound ϵ. We refer to the tuple (δ, ϵ) as the *contract* between controller and estimator. The estimator then provides an estimate that respects the contract. By requesting estimates with varying contracts during system operation, the controller is able to adapt the closed-loop system performance in real-time according to the current condition 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-and-estimation algorithm. A high-level view of this setup is shown in Fig. 1.

To ensure that the estimator can respect the contract (alternatively, that the controller is only requesting contracts that can be fulfilled by the estimator), the estimator is profiled off-line. Namely, the estimator's parameters are varied and for each setting of the parameters, it is run on a *profiling data set*. This yields a finite set of (δ, ϵ) values, each one corresponding to a 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), represented by the set Δ. Examples of such a curve are shown in Figs. 7 and

8. The detailed procedure for obtaining such a curve for a perception based algorithms is given in Section VI.

At run-time, when the estimator receives a (δ, ϵ) contract request from the controller, it can adapt its execution paths to respect the contract, namely, to provide a state estimate in real-time within the requested error bound ϵ, and within the requested deadline δ.

In addition, the controller is designed with the knowledge of the error-delay curve of the estimation algorithm, and requests contracts from that curve. Thus, the error-delay curve constitutes the *interface* between controller and estimator. This gives the controller the ability to leverage the flexible nature of the estimation algorithm to maximize some performance measure of control performance.

Fig. 4 shows the closed loop architecture in a system with co-design of the estimator and controller. In the co-designed system as presented in this paper, 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 co-design architecture are a contract based perception-and-estimation algorithm, a robust control algorithm that computes an input to be sent to the physical system being controller as well as the operating mode for the contract time estimator, and the interface between them. More details on these components are in the following sections.

A. *Contract based perception algorithms*

A contract based perception-and-estimation algorithm can operate at different deadlines and provide a usable solution for the control algorithm to operate on. This flexible operation is achieved by composing the algorithm of functional blocks that have different execution times and result in different qualities of outputs.

An example is a Computer Vision (CV) based Object recognition algorithm which is composed of different functional blocks of varying execution time which result in a different accuracy when linked together to provide the functionality of an object recognition algorithm. E.g. the pixel classifier in the first stage of such a CV algorithm could be a Gaussian Mixture Model with 2, 4, or 6 components, with more components providing better classification performance (over-fitting is ruled out by cross-validation) at the cost of more computation time. Functions with similar characteristics like example above, when profiled extensively offline and composed in the right order at run-time can be used to compose a contract time anytime perception and estimation algorithm. More details follow in section VI.

B. *Interface between contract based perception and robust control*

For the control algorithm to be able to leverage the flexible nature of the contract based perception algorithm, it must have information about the computation time versus output quality trade-off that the contract based perception algorithm offers. An interface that achieves this is obtained by representing the profiled behaviour of the contract based algorithm to varying deadlines, as points on a perception quality versus deadline (δ, ϵ) curve, e.g. in Fig. 7. With this profiled curve available to the controller at runtime, the exchange of information between the contract based perception-and-estimation algorithm and the control algorithm consists of the controller assigning a deadline (δ), or a contract to the perception algorithm while expecting a bound on the error (ϵ) of its output. The perception algorithm then returns an output after internally deciding the composition to best meet the deadline and the expected quality requirement. Through extensive offline profiling, we guarantee with a high degree of confidence that the contract based estimator does not violate the contract.This in particular helps in formulating a control algorithm that provides mathematical guarantees on the feasibility of constraints for the safe operation and stability of the closed loop dynamic system as is covered in section V.

C. *Robust Control with contract based perception algorithm*

The control algorithm is designed to pick the best operating point for the estimator, or the right (δ, ϵ) contract to request from the perception and estimation algorithm. This is done based on the current state of the physical system to maximize a performance measure while being robust to the varying computation time and the varying estimation errors of the estimator with different contracts as is provides estimates to the controller. In section V we present a control algorithm that achieves this while also guaranteeing feasibility of system constraints the stability of the closed loop system.

IV. Robust Control with Contract-Based Estimator

In this section we present the mathematical formulation to model the controller and physical system from Fig. 1, and demonstrate how the controller can, in real-time, use knowledge of the estimator's error-delay curve to decrease computation delay and power in an error-aware fashion.

A. *System Model*

Consider a hexrotor, which is an autonomous flying robot with six rotors, shown in Fig. 3. The state x of the hexrotor is made of its 3D position and 3D velocity. The input u to the robot consists of the desired pitch and roll angles, and the desired thrust. The hexrotor's mission is to fly a pre-defined pattern 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 and approximated by the following Linear Time-Invariant (LTI) ODE:

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

where $x \in \mathbb{R}^n$ is the state constrained to lie in a set $X \subset \mathbb{R}^n$, $u \in \mathbb{R}^m$ is the control input constrained to lie in a set $U \subset \mathbb{R}^m$, and $w_c \in \mathbb{R}^n$ is the bounded process noise assumed to lie in a set $\mathcal{W}_c \subset \mathbb{R}^n$. $A_c \in \mathbb{R}^{n\times n}$ and $B_c \in \mathbb{R}^{m\times n}$ are matrices. LTIs model a wide range of systems, and our

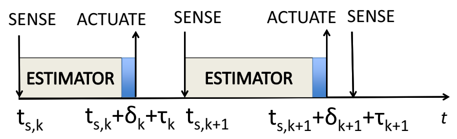

Fig. 5. Time-triggered sensing and actuation. The figure shows the varying execution time for the estimator and the blue area shows the execution time for the controller, which is small.

results apply to arbitrary LTIs of the form given in (1) with compact and convex constraint sets X, U and $\mathcal{W}_c$. The sets X and U are part of the problem statement and are either chosen by the designer or determined by physical constraints of the physical plant and the actuators. For the hexrotor, X captures limits on the state such that the LTI dynamics provide a good approximation of the true nonlinear dynamics. The set U restricts the inputs to values that can be supported by the rotors.

B. Time-Triggered Sensing and Actuation

For flight the hexrotor needs to determine its current position and speed, i.e., it needs to produce an *estimate* of its current state x. It does so by taking a video during flight through a downward facing camera, detecting and tracking features across frames, and deducing its own position relative to these features. The camera captures a new frame every $T > 0$ seconds, thus resulting in periodic measurements at instants $t_{s,k} = kT$, where $k \in \mathbb{N}$.

The sampled measurement is fed to the estimator that computes the state estimate $\hat{x}_k := \hat{x}(t_{s,k})$ with the desired accuracy ϵ_k *determined by the controller in the previous time step*. The controller then uses this state estimate to compute the control input u_k as well as decide on the state estimate's delay and accuracy contract $(\delta_{k+1}, \epsilon_{k+1})$ for the next step. This control is 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. 5.

In our setting, the controller has access to the delay-error curve Δ of the estimator, and makes contract selections *from that curve*. This curve is obtained offline as explained in Section III, and exemplified in Section VI. We remark that in each step $k \geq 0$, the estimation accuracy ϵ_k and hence the delay δ_k are already decided in the previous step and known to the controller. In the first step $k = 0$, the initial accuracy ϵ_0, the initial delay δ_0, and the initial control input u_{-1} are chosen by the designer.

C. Control Performance

The goal of the controller is twofold: it needs to ensure that the reference pattern is adhered to as closely as possible, and that the energy consumed to fly this pattern is minimized. Thus we may 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 matrices. Second, $\pi(\delta)$ captures the average power consumed to perform an estimation of duration δ. This power information is collected offline during the estimator profiling phase. The paper's formulation holds for much more general stage cost functions. These stage cost functions are chosen by the designer to achieve a desired control performance.

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

D. Discretized Dynamics

Because of time-triggered sensing and actuation, from time $t_{s,k}$ to $t_{a,k}$, the previous control input u_{k-1} is still used. Then at $t_{a,k}$ the new control input u_k is computed and applied by the controller (see Fig. 5). The discretized dynamics are given by

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

in which

$$A = \mathrm{e}^{A_c T}, \quad w_k = \int_0^T \mathrm{e}^{A_c(T-t)} w_c(t_{s,k} + t) dt$$
$$B_1(\delta) = \int_0^\delta \mathrm{e}^{A_c(T-t)} B_c dt, \; B_2(\delta) = \int_\delta^T \mathrm{e}^{A_c(T-t)} B_c dt.$$

Here w_k is the accumulated process noise during the interval, and is constrained to lie in a compact convex set $\mathcal{W}$ because $w_c(t)$ lies in the compact convex set $\mathcal{W}_c$ and T is finite. Note that both the current control u_k and the previous control u_{k-1} appear in (2). Furthermore, the input matrices $B_1(\delta_k)$ and $B_2(\delta_k)$ depend on the delay δ_k. The estimation accuracy ϵ_k affects the state estimate $\hat{x}_k$ used by the controller to compute u_k; therefore ϵ_k indirectly affects the dynamics via the control input.

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-based setup of Fig. 4. The mathematical details and derivations are available in the online technical report [15]. 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-based control and estimation framework as presented in Section III and Fig. 4. 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 two things: first is the control input u_k to be applied to the physical system at time $t_{a,k}$. The second is 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 quantifies the importance of power consumption to the overall performance of the system.

The contract-based 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.

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 **Robust Adaptive Model Predictive Control (RAMPC)** seeks to solve the following optimization problem which we denote by $\mathbb{P}_\Delta(\hat{x}_k, \delta_k, \epsilon_k, u_{k-1})$:

$$J^*[0:N] = \min_{\mathbf{u},\mathbf{x},\underline{\delta},\underline{\epsilon}} \sum_{j=0}^{N} \left(\ell(x_{k+j}, u_{k+j}) + \alpha\pi(\delta_k)\right) \qquad (3)$$

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 $J[0:N]$. In the remainder of this section we discuss how to make this problem tractable. As in regular MPC [16], 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, i.e. $(\delta_{k+j}, \epsilon_{k+j}) = (\delta, \epsilon)$ for all $1 \leq j \leq N$. 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 [17], [18]. 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, 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 [15]. 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

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

Algorithm 1 summarizes the RAMPC algorithm.

Algorithm 1 Robust Adaptive MPC 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:         $J^*_{(\delta,\epsilon)} \leftarrow$ 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
```

We prove the following result in the technical report [15]:

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 Alg. 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. Contract based perception algorithms

In Section III, we postulated the existence of an Estimation Error vs Computation Delay curve Δ. This curve is used at every time step by the controller to determine the operating point (δ, ϵ) for the next time step. In this section we demonstrate in detail how such a curve may be obtained for particular applications and how points along the curve are realized at runtime by the contract based perception algorithms.

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

The first step towards profiling a contract-based estimator is to identify the individual components (or algorithms) of the perception tool chain. The second step is to identify parameters of each component, such that modifying the values of these parameters leads to a change in the execution time and accuracy of the component's output. This may be as simple as changing the number of iterations in a loop [12] or finding alternate implementations with different resultant execution times δ and estimation error ϵ. We call these parameters *knobs* of the component. The tool chain We implement this procedure on a Computer Vision (CV)-based

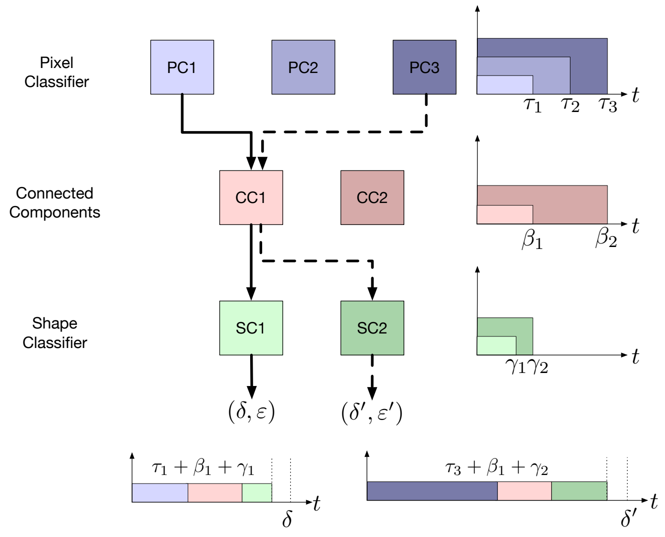

Fig. 6. Illustration of the various components used to compose the contract based perception algorithm and their representation as real-time tasks. For a given (δ, ϵ) contract, knob settings are chosen at run-time resulting a schedule to execute these sequential components, or tasks to respect the contract.

object recognition tool chain. An overview of the tool chain is shown in Fig. 6.

The CV tool chain takes in a video stream and tracks an Object Of Interest (OOI) across the frames. The first stage of the chain is a pixel classifier that assigns to each pixel of the image (after potential pre-processing) the probability of its being a pixel of interest, i.e., of belonging to an OOI or being a part of the background. A binary image is then obtained which assigns the value 1 to pixels of interest, and 0 to all others. Next, filtering and a Connected Components (CC) algorithm is run on the binary image to get rid of noise in the classification process and segment its 1-valued pixels into disconnected objects. A shape classifier is then run on each object to determine whether it is an object of interest or not.

In our implementation, the pixel classifier is a Gaussian Mixture Model (GMM) classifier, whose knob is the number of components in the GMM. Fewer Gaussians in the GMM yield a faster but less accurate classifier while more Gaussians will result in a higher execution time but provide better classification performance. Knob values that cause data overfit are discarded by a cross-validation stage as is standard.

The filtering and Connected Components algorithm are lumped into one stage and have a two-valued knob to choose between a 4-connected and 8-connected component implementation. The shape classifier is also a GMM, but the knob for it is the number of shape features (like eccentricity and lengths of major and minor axes). In our experiments the number of knob settings for the entire chain is K = (#Gaussians for pixel classifier, #neighbors for CC, #features for shape classifier), and has a total of $3 \times 2 \times 2 = 12$ values.

Note that for any given component in the chain, the relation between knob value and quality of output is not

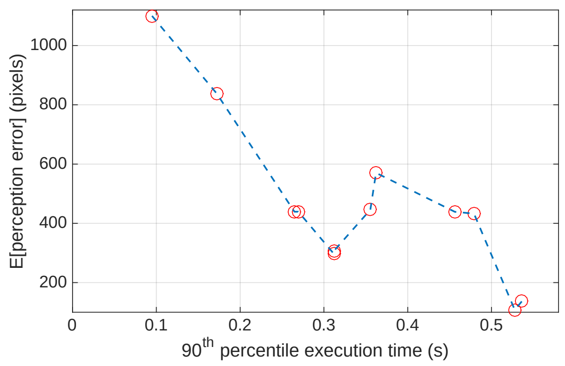

Fig. 7. Profiled delay-error curve for the object detection tool chain run at different parameter settings.

necessarily monotonic. The pixel and shape classifiers are machine learning algorithms that need to be trained on a training set before being used and like all machine learning algorithms, their output quality for a given knob setting will depend on the actual data set. The same is a fortiori true of the quality of the output of the entire chain. This is also reflected in Fig. 7 which shows the mean perception error[1] and the 90^{th} percentile execution time for the different knob settings.

The final step is to profile all the possible combinations of knobs by running the tool chain on a test data set. This profiling gives us: a) the output quality (or accuracy) of the perception-and-estimation tool chain under consideration, and b) information about execution times for the stages of the perception tool chain under different knob settings. This information gathered offline is useful for making decisions at run-time. Fig. 7 shows the profiled performance of the CV tool chain.

B. *Run-time execution of the contract based perception algorithm*

After the contract based perception algorithm has been composed and the execution time distributions of its individual components and the quality distributions from composing together various knob settings have been profiled, we can make run-time decisions for knob settings in order to realize a given (δ, ϵ) contract.

This is the equivalent of selecting different versions of tasks (knobs for stages) and scheduling them in sequential order to best perform the object recognition task while maximizing utilization (or estimation accuracy) and meeting the given time contract or deadline. Fig. 6 shows the different task versions for each knob in the different stages and the resulting schedule based on the knob settings for the stages. The offline profiling allows us to set the knobs such that we can achieve a feasible schedule for the given deadline, δ while maximizing the utility, or the expected accuracy of the perception algorithm.

[1]Error is the distance between the true centroid and the estimated centroid of the OOI

C. Visual Odometry

Another algorithm we consider and later use in Section VII is the Semi-Direct Monocular Visual Odometry (SVO) [19]. The visual odometry algorithm detects corners in an image and tracks them across video frames to perform self-localization of a moving robot. These estimates are used in the closed loop control system that flies the hexrotor, hence it is important for the visual odometry to run at or faster than frame rate in order to provide a timely state estimate to the control algorithm The number $\#C$ and quality of corners detected in a frame directly affects the runtime of the corner detector and the resulting quality of the state estimate. Generally speaking, detecting more corners requires a longer runtime, and results in better self-localization as long as we are analysing a feature rich scene, i.e., *assuming acceptable quality of the detected corners*. Thus the number $\#C$ of corners is a knob which can be varied to obtain an error-delay curve for self-localization with the visual odometry algorithm. If the scene is not rich enough in features, and a sizeable fraction of the $\#C$ corners are of poor quality (i.e., unstable or hard to track across frames), then we can expect the self-localization error to actually increase as the poor quality of the unstable corners detected adds noise to the visual odometry estimates.

Fig. 8 shows the error-delay curve of self-localization error using the SVO. The curve was obtained on an Odroid-U3 [20], which is the same processor as the one used on the hexrotor for on-board computation. For each value of the knob $\#C$ (i.e., each requested number of corners), we ran the visual odometry algorithm on a video sequence recorded by the downward facing camera on the hexrotor while flying certain pre-set patterns. Ground truth for computing the self-localization error was obtained using a Vicon motion capture system which provides position estimates with better than millimeter level precision. As we repeat each flight several times, this results in a distribution of (δ, ϵ) values for each value of $\#C$. We retained the 90^{th} percentile values for δ and ϵ, since these can be used as the worst-case estimates and delays by the controller of Ssction IV. It can be seen that a larger number of requested corners produces a smaller estimation error and longer runtime. Starting at 250 corners, the error increases, however. We hypothesize this is due to the decreasing quality of the corners being returned by the corner detection algorithm.

VII. Case Study: Real-time feedback control of a hexrotor with contract based estimation and robust control

A. Experimental setup

To evaluate our methodology on a real platform, we applied it to a hexrotor tasked with repeatedly following a given circular trajectory. We use SVO (Section VI) as estimator and RAMPC as the controller. Details of the experimental setup, and of the obtained delay-error curve for SVO, are in the Appendix.

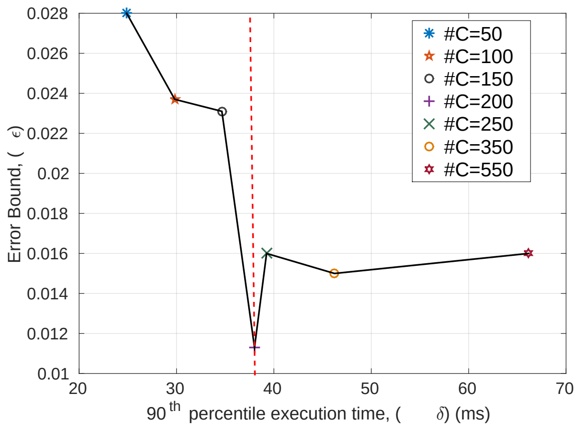

Fig. 8. Error-delay curve for the SVO algorithm running on the Odroid-U3 with different settings of maximum number of features ($\#C$) to detect and track. The vertical line shows the cut-off for maximum delay and the SVO settings that are allowable for closed loop control of a hexrotor at 20Hz.

B. Experimental Evaluation

After profiling the performance of the perception and estimation algorithm and formulating the Robust Adaptive MPC controller for the hexrotor linearized around hover and modelled as an LTI system (Eq. 1), we experimentally evaluate the tracking performance and estimated energy consumption based on actual flights around a pre-defined trajectory. For comparison, we use a Model Predictive Controller with the same cost function and initial feasible sets as in our Robust MPC formulation. The MPC controller is an appropriate baseline against which to measure the benefits of our co-design method, as it is a similar control algorithm that does not leverage co-design and is unaware of the estimator algorithm that gives it a state estimate.

For the evaluation, we fly in a predefined circular trajectory, repeating the experiment 10 times to gather enough data to conclusively measure the performance of RAMPC for different values of α and MPC with fixed modes of (δ, ϵ). Note that since the controller was a sampled discrete-time controller working with simulated 20Hz camera updates, this realistically restricts us to using modes of estimator operation with delay δ less than 1/20s, or one sampling period, i.e. modes corresponding to 50, 100, 150 and 200 maximum corners from the FAST detector (see Fig. 8). These modes and their estimated power consumption is in Table I. Note, #C represents the maximum number of FAST corners requested, ϵ shows the worst case error bound on the state estimate, δ is the 90^{th} percentile execution time for that mode, and P represents the expected power consumption in that mode as profiled offline. This power consumption is the computation power used by a particular mode in excess to the idle power for the Odroid used for profiling, which was 1.5W.

C. Experimental Results

Once the flights are complete, to get a more accurate picture of how the controllers really performed, we use the

TABLE I

ESTIMATION MODES USED IN THE EXPERIMENT.

Mode	#C	ϵ	δ (ms)	P(W)
0	50	24.88	0.028	0.778
1	100	29.82	0.0237	0.862
2	150	34.66	0.0230	0.870
3	200	38.01	0.0113	0.951

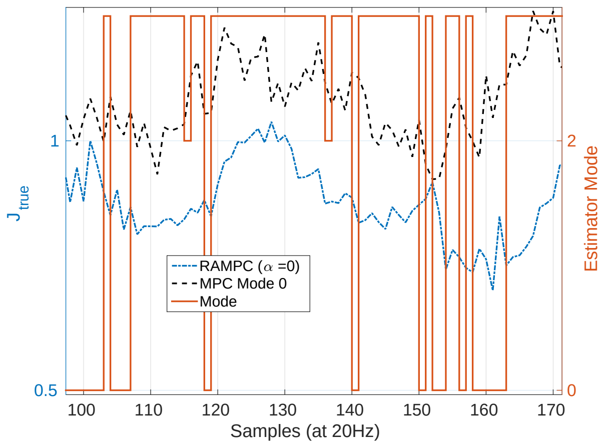

Fig. 9. Tracking cost at each time step for MPC (fixed mode 0 estimator) and RAMPC with $\alpha = 0$. Note how the RAMPC performs better (lower cost) than the MPC and there is dynamic switching of estimator modes at runtime leading to improved performance for the RAMPC.

following function to measure tracking performance at each time step.

$$J_{true}(t) = (x(t) - x_{ref}(t))^T Q(x(t) - x_{ref}(t)) + u(t)^T Ru(t) \quad (5)$$

Note that since we have access to the true position and velocities ($x(t)$) of the hexrotor with the Vicon system, we can obtain the true tracking cost. Table II shows the mean of the above function over the 10 flights for both MPC across all fixed modes and RAMPC with different values of α. It also shows the estimated energy consumption based on the time spent in each mode (which can be seen in Table III for RAMPC). RAMPC shows better tracking performance (lower mean J_{true}) than MPC in all cases, except for $\alpha = 0.2$, thus demonstrating the improved control performance that can be obtained by dynamically switching between estimation modes in-flight at runtime.

Fig. 9 shows how the tracking cost (J_{true}) evolves over time for RAMPC (with $\alpha = 0$) and MPC (fixed mode 0) for a portion of the hexrotor flight. The estimator modes selected by RAMPC are overlaid in orange. Fig. 9 demonstrates that RAMPC has uniformly lower tracking cost than MPC, enabled by RAMPC's dynamic switching of estimator modes at runtime. Note that RAMPC exhibits better tracking performance throughout the flight and not just in this portion, and also outperforms MPC at other modes (see Table II).

Figure 10 shows that RAMPC provides better tracking

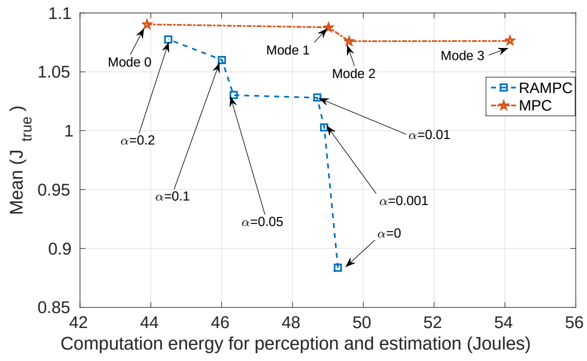

Fig. 10. Tracking cost vs estimated computation energy for executing the perception and estimation algorithm. Depending on which fixed mode of the estimator is chosen, MPC operation consumes a different amount of energy. Using RAMPC as the controller, the different energies are due to different runtime scheduling of estimator modes based on based α. It is worth noting that RAMPC with co-design outperforms standard MPC on tracking performance across the entire range of energy consumption.

performance while using less energy to do so. For any fixed energy budget (a point on the x-axis), RAMPC delivers lower tracking cost (y-axis) than MPC. While MPC's tracking error is relatively constant across modes, RAMPC is able to balance tracking error with energy consumption by varying the α parameter. RAMPC's switching between estimation modes improves not only the control performance but also energy efficiency.

TABLE II

TRACKING PERFORMANCE AND COMPUTATION ENERGY

Controller	Est. Mode/ α	$E[J_{true}]$	$\sigma(J_{true})$	$Energy(J)$
MPC	0/ –	1.0903	0.104	43.89
MPC	1/ –	1.0878	0.087	49.02
MPC	2/ –	1.0760	0.098	49.60
MPC	3/ –	1.0762	0.088	54.15
RAMPC	–/0	0.8836	0.079	49.28
RAMPC	–/ 0.001	1.0029	0.093	48.90
RAMPC	–/ 0.01	1.0280	0.089	48.69
RAMPC	–/ 0.05	1.0302	0.096	46.33
RAMPC	–/ 0.1	1.0601	0.086	46.01
RAMPC	–/ 0.2	1.0776	0.083	44.49

Fig. 11 shows the degradation (increased mean J_{true}) in tracking performance and reduction in energy consumption as the weight α for the computation power in the cost function is increased. As energy becomes more important, RAMPC smoothly balances tracking cost and energy consumption. Table III quantifies how RAMPC makes this trade-off, by showing the fraction of time spent in the 4 modes with RAMPC as α changes. While time is split between modes 0 and 3 with $\alpha = 0$, more and more time is spent in the low-power (but less accurate) mode 0 as α increases.

VIII. CONCLUSION

In this paper we presented a contract-based methodology for co-design of estimation and control for autonomous sys-

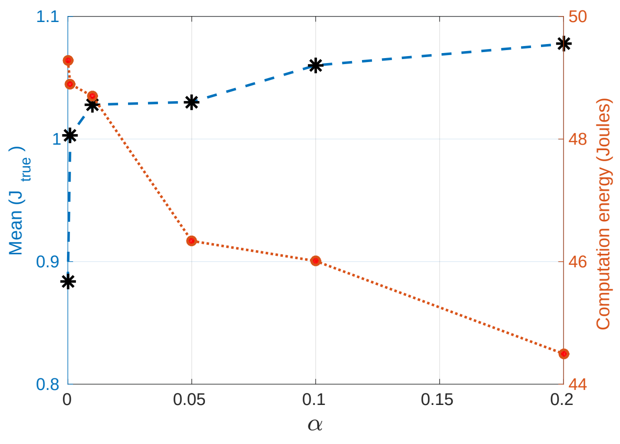

Fig. 11. RAMPC tracking cost and estimated computation energy for the perception and estimation algorithm as a function of α.

TABLE III

FRACTION OF TIME SPENT IN DIFFERENT ESTIMATOR MODES AS α CHANGES FOR RAMPC

α	Mode 0	Mode 1	Mode 2	Mode 3
0	0.461	0.009	0.020	0.510
0.001	0.494	0.001	0.029	0.467
0.01	0.512	0.005	0.039	0.444
0.005	0.692	0.000	0.156	0.152
0.1	0.691	0.000	0.218	0.091
0.2	0.897	0.000	0.098	0.005

tems. 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 maximise 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-based 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 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 co-design. 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. Another focus is on an automated tool chain to profile perception algorithms commonly used in autonomous systems.

ACKNOWLEDGEMENTS

We would like to thank Kuk Jang for his help in creating several of the diagrams in this paper.

REFERENCES

[1] M. Boddy and T. Dean, "Solving Time-dependent Planning Problems," *Joint Conf. on AI*, pp. 979–984, 1989.

[2] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI Magazine*, vol. 17, no. 3, 1996.

[3] 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, 2008.

[4] M. Wellman and C. L. Liu, "State-Space Abstraction for Anytime Evaluation of Probabilistic Networks," *Conf. on Uncertainty in AI*, 1994.

[5] R. Mangharam and A. Saba, "Anytime Algorithms for GPU Architectures," in *Proc. of the IEEE Real-Time Systems Symposium*, 2011.

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

[7] R. Bhattacharya and G. J. Balas, "Anytime control algorithm: Model reduction approach," *Journal of Guidance and Control*, vol. 27, no. 5, pp. 767–776, 2004.

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

[9] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal analysis of timing effects on closed-loop properties of control software," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*, Dec 2014, pp. 53–62.

[10] D. de Niz, L. Wrage, N. Storer, A. Rowe, and R. Rajukar, "On Resource Overbooking in an Unmanned Aerial Vehicle," *IEEE/ACM Third International Conference on Cyber-Physical Systems*, 2012.

[11] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Mgmt." *IEEE RTSS*, 1997.

[12] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 124–134. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025133

[13] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &#38; Applications*, ser. OOPSLA '13. New York, NY, USA: ACM, 2013, pp. 33–52. [Online]. Available: http://doi.acm.org/10.1145/2509136.2509546

[14] R. St. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 505–516. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665746

[15] Y. V. Pant, K. Mohta, H. Abbas, T. X. Nghiem, J. Devietti, and R. Mangharam, "Co-design of anytime computation and robust control (supplemental)," Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, Tech. Rep. UPenn-ESE-15-324, May 2015, http://repository.upenn.edu/mlab_papers.

[16] E. Camacho and C. Bordons, *Model predictive control*. Springer Verlag, 2004.

[17] A. Richards and J. How, "Robust model predictive control with imperfect information," in *American Control Conference*, 2005, pp. 268–273.

[18] 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, 2001.

[19] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 15–22.

[20] "ODROID-U3," http://odroid.com/, accessed: 2015-05-13.

[21] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[22] J. Mattingley and S. Boyd, "Cvxgen: a code generator for embedded convex optimization," *Optimization and Engineering*, 2012.

[23] "ODROID Smart Power," http://odroid.com/, accessed: 2015-05-13.

Appendix

1 To evaluate our methodology on a real platform, we applied it to a hexrotor with the Odroid-U3 as a computation platform, running the Robot Operating System (ROS) [21] in Ubuntu. For the evaluation, the hexrotor is tasked with repeatedly following a given circular trajectory. As can be seen in Fig. 12, the visual odometry algorithm can occcasionaly take a long time to give a pose estimate. In our formulation in section V we have assumed that the estimator satisfies the (δ, ϵ) contract requested by the controller. Thus, to ensure that the estimator fulfills the contract and that the mathematical guarantees provided by our RAMPC formulation hold, instead of using the visual odometry algorithm to fly the robot, we injected delays and errors into the measurements from Vicon, which is a high accuracy localization system. These delays and errors were selected from the Δ curve obtained by profiling the SVO algorithm (see Section VI-C). The hexrotor flies using these pose estimates and our RAMPC Algorithm for both the position control and setting the time deadline for the next estimate. The RAMPC has the positions and velocities in the 3-axes as its states (x), and generates control inputs in the form of desired thrust, roll and pitch (yaw is set to 0) in order to compute a given reference $x_{ref}(t)$ for a low-level controller to track. The RAMPC is coded in CVXGEN [22] and the generated C Code is integrated in the ROS module for position control of the hexrotor, running at 20Hz. The sets $\mathcal{Z}_j$ are done offline in MATLAB and then used in CVXGEN as Polyhedron type constraints. The constraint set X defines a safe set of positions and velocities in the flying area. The constraint set U of inputs keeps desired pitch and roll magnitudes less than 30 degrees and desired thrust within limits of the hex-rotor abilities. Later in this section we show that our approach dynamically schedules different modes of the contract-based perception and estimation algorithm at runtime and also controls the dynamical system in an energy-efficient way while providing good tracking performance. In the evaluation subsection we will compare our results to a baseline Model Predictive Control algorithm that does not leverage co-design and operates at a fixed (δ, ϵ) mode of the perception/estimation algorithm.

A. Profiling the perception and estimation pipeline

Recall that in section V, the control algorithm needs the profiled Δ curve for the estimator. In our experimental setup, the estimator is given by the SVO algorithm. Fig. 8 shows the bound on estimation error and the 90^{th} percentile execution times. This is obtained by varying the maximum number of corners knob in SVO, denoted by $\#C$, and flying extensively over a relatively feature-rich environment for each value of the knob (Fig. 3). The estimation error is computed using ground truth position obtained through the Vicon motion capture system. We profiled the performance for the same trajectory with different settings of the odometry offline by logging images and IMU data in-flight, and then running the visual odometry code on the Odroid-U3 offline. This accurately recreates the in-flight environment that is present for the visual odometry algorithm and this profiling is then used online for making in-flight decisions by the control algorithm.

Also needed for the control optimization in Eq. 4 is a measure of the power consumption by SVO at different values of the knob $\#C$. Power measurements are made using the Odroid Smart Power meter [23], which measures consumption at 10Hz to milliwatt precision. To avoid the physical challenges of fitting the power meter onto our hexrotor platform, we measure the power consumption of the Odroid board on the ground, running the same controller and vision workloads as it does during flight as explained above, and at different knob settings. We measure the power consumption of the entire Odroid board, including CPU and DRAM power consumption. Since the profiling of power is done offline with other peripherals plugged into the odroid (e.g. a monitor and keyboard), we measure the idle power of the Odroid and subtract that from the power measurements when the SVO algorithm is running on it in different modes. This gives us a more accurate measure of the workload that the visual odometry task is responsible for. This offline profiling now allows us to formulate the co-design problem for the hexrotor and experimentally evaluate our methods.

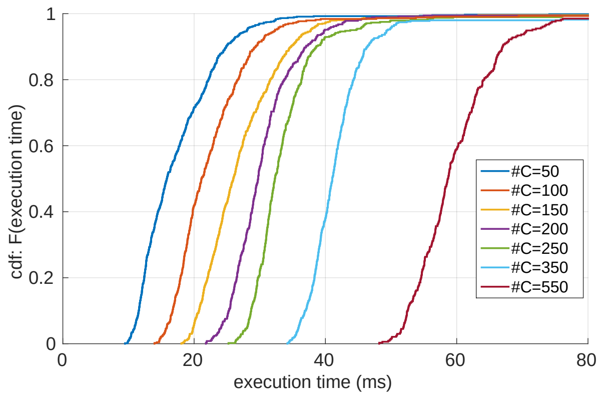

Fig. 12. Cumulative distribution of profiled execution times for visual odometry running on the Odroid-U3 for varying maximum number of corners from the SVO algorithm.