Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

Specifying and Verifying Visual Grasping Tasks

Eugenia Shkel and Nicola J. Ferrier
Department of Mechanical Engineering
University of Wisconsin, Madison, WI, 53706
eugenia@cae.wisc.edu, ferrier@engr.wisc.edu

Abstract

This paper presents an approach to the specification of requirements, and verification of design, for a robot or other intelligent system. The approach is demonstrated on a typical robotic task – visual grasping. Formal mathematical reasoning is used to show that a design conforms to the system requirements. Typically the requirements define safety and functionality constraints on the system and components. Formal analysis allows the system designer to evaluate the system behavior and verify the system parameters that guarantee safe and robust system performance.

1 Introduction

With a growing number of computer-controlled systems we face a major challenge: development of reliable, robust, and safe real-time intelligent systems. Intelligent autonomous system development can be decomposed into three phases: system modeling, requirements specification, and behavior verification [7, 19, 6]. Robotic systems are usually complex, hierarchical, and physically distributed. They must deal with inconsistent, incomplete, and delayed information from various sources. To address these problems, this paper demonstrates suitable analytical tools which would allow a designer of a complicated robotic system to use a formal mathematical reasoning when checking that a design conforms to the requirements. A methodology for analyzing such complex systems would enable a design engineer to refine the control structure for optimal performance given specific system dynamics and limitations. Formal verification tools have proven to be important in the development of real-time software [3]. We hope to develop and apply these techniques to robotics resulting in a more efficient, systematic, and reliable robotic system design process.

Numerous mathematical models and formalisms have been proposed over the past years for computer controlled systems. Timed Petri nets[4], constraint nets[23], hybrid automata[2], and timed $\forall$-automata[14] have been developed for the specification and verification of concurrent programs and hybrid systems. Discrete event system theory[18] has been applied to the synthesis of supervisory control. Logical models of continuous state systems have been described in [5, 12, 16, 22]. These intelligent system models are currently in the development stage: no model has reached a level of universal acceptance. The adaptation of these theoretical techniques for real world applications is a subject for future investigation.

In the following sections we present a formal verification of a visual grasping task using the techniques of duration calculus [7, 19, 6]. Duration calculus (DC) is an extended temporal or interval logic [1, 15], which can be used to specify and reason about real-time and logical constraints in dynamical systems without the explicit mention of time instants. Duration calculus has already been used to design a gas-burner [7, 19], a railway crossing [21], and a digital controller of hydraulic arm manipulator [20]. We apply DC to a complex system, a robot performing visual grasping. DC is used to describe state durations, progress from state to state, and stability of the robotic system states. The task of visual grasping (or robotic catching) is of interest in both manufacturing, and in tele-operation applications [8, 17]. Although there are numerous papers that demonstrate success at this task [10, 9], each working system involved fine tuning of parameters highly specific to the particular project. In this paper we are trying to analyze the task of visual grasping in the framework of system specification and verification. It is assumed that our system has bounded, but unknown, control parameters and delays. In modeling an intelligent system the challenge is to be able to analyze each level of abstraction in detail and the overall behavior of the robotic system in general. Formal methods are used to derive sufficient and necessary conditions on the control and delay parameters, such that the entire system satisfies the given duration calculus requirements. This explores the task at a meta-level, considering the effect of various communication and computation times, system dynamics, etc. on the ability to successfully complete a visually

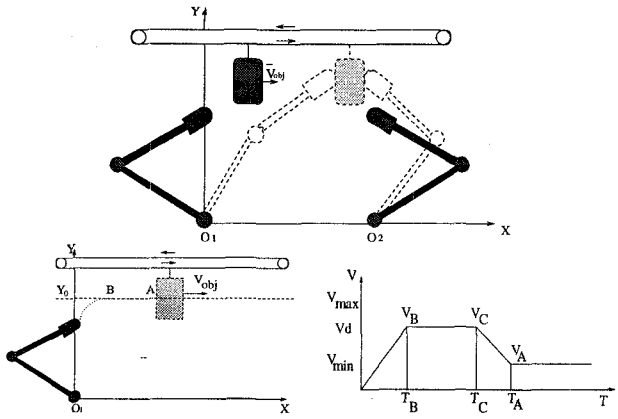

Figure 1: Grasping the object.

guided grasping task. The design trade-offs that others had to make in designing a grasping system can be made explicit within our model.

The task of grasping is often decomposed into phases, such as *Idle, Approach, Pregrasp, Grasp* (e.g. [8]). The transition between phases is triggered by a control command or by sensor feedback. For a simple task such as grasping, analysis of the control flow through these phases leads one to believe that the task can be executed: if each phase is successful and the transitions between phases are successful, then the task should be successful. If one imposes constraints on the task, such as an upper bound on the time to complete the task, or if particular hardware imposes constraints, then analysis of the individual phases is not sufficient to analyze the system as a whole. Often a robotic system is "fine tuned" to perform a specific task given particular constraints, however a change in hardware may lead to controller re-design. Formal techniques make this tuning more systematic.

With a visual grasping task, various questions may arise, such as: 1) how much chatter at contact is permitted? 2) if the object to be grasped is moving at speed v_o and the robot hand has a maximum speed, v_m, can the robot catch the object? 3) how fast can the robot grasp an item? and 4) will the robot do what it is designed to do? Answers to such questions will depend on the individual hardware and software parameters. We are interested in the interaction of the various control/hardware and software parameters on each level of control and finding a specification language, which would allow us to reason on the level of differential equations as well as at the logical level.

2 Visual grasping.

To demonstrate the technique of duration calculus we consider the task of grasping a moving object with a planar two-fingered hand using visual information about the object and the assumptions:

- each finger is equipped with tactile and force/torque sensors;
- the size, shape, and speed of the object are not known but are computed from visual information.
- the moving object is constrained (e.g. by a conveyor) and has translational speed v_{obj}.

Under these assumptions our goal is to design a *provably* reliable autonomous control system which will grasp the object using only information about object geometry obtained from the camera and using the tactile sensors to detect a contact with the object.

2.1 Task description.

Coordinates of the boundaries of the object are obtained by processing an image of the object, which is assumed to take time δ. Assume A is a point midway along the left boundary of the object and define the line $y = \frac{y_2 - y_1}{2} = y_0$, where y_1 (y_2) is the coordinate of the upper (lower) bound of the object. The left fingertip is required to reach A as fast as possible and attain velocity v_{min} at point A (fig. 1). Velocity v_{min} is the transition velocity to the phase *Contact*. The *Approach* phase can be divided onto three sub-phases:

1. *Approach B.* Optimally approach the line $y = y_0$ with the velocity v_d at point B. v_d is constrained by v_{max}, and v_{min}, and depends on the distance from B to A, and maximum deceleration capability of the system.
2. *Approach C.* Move with constant speed v_d along the line $y = y_0$ from B to C, where C is a starting point of deceleration to a low transition velocity v_{min}.
3. *Approach A.* Deceleration from velocity v_d to velocity v_{min} attained at location A. Because the object is moving along the line ($y = y_0$), the coordinates of the point A can be calculated as $(x_A, y_A) = (x_0 + v_{obj}t, y_0)$, where x_0 and y_0 are x- and y-coordinates of the point A at the starting position. The position of the fingertip (x_F, y_F) can be calculated using inverse kinematics and shaft encoder angle information. The expected-end-event (reach A) is detected when the coordinate of the fingertip x_F is equal to the coordinate x_A of the point A. Thus, the event $(x_F = x_A)$ indicates switching to the next control phase *Precontact*.

In phase *Precontact* (fingertip is moving with velocity v_{min} and location A has been achieved with sufficient accuracy), the fingertip force is monitored, and when the force exceeds a prescribed level, the expected-end-event *Contact* is declared.

When the event *Contact* is recognized, the control system is switched into a different control phase which maintains contact with some desired force. The finger and object move with the same velocity v_{obj}, until contact with another finger is sensed. Detecting contact

$\lceil\ \rceil$	holds on point intervals $[b,b]$
$\lceil \mathtt{Q} \rceil$	holds on $[b,e]$ $(b<e)$, if Q has value *true* almost everywhere in $[b,e]$
$\Diamond\mathcal{D}$	holds on $[b,e]$ if $\mathcal{D}$ holds on some subinterval of $[b,e]$
$\Box\mathcal{D}$	holds on $[b,e]$ if $\mathcal{D}$ holds on any subinterval of $[b,e]$
$\mathcal{D}_1 \rightarrow \mathcal{D}_2$ ($\mathcal{D}_2$ follows $\mathcal{D}_1$)	holds if $\mathcal{D}_2$ holds on some subinterval from the point (if any) where $\mathcal{D}_1$ ceases to hold.
$\mathcal{D}_1 \sim t \rightsquigarrow \mathcal{D}_2$ ($\mathcal{D}_1$ leads to $\mathcal{D}_2$ in t)	holds on an interval with $\mathcal{D}_1$ holding on an initial subinterval if $\mathcal{D}_2$ starts holding within time t

Table 1: Duration Calculus Notational Semantics.

from both fingers will trigger a switch into phase *Lift*, followed by the phase *Move* and so on.

2.2 System requirements

For reliable performance of the autonomous system, it is required that system should be self-controlled: if an error in the system occurs, the controller design should guarantee that the error will be detected within a certain time interval and the system will safely recover from the critical state. Specifically, for the task of grasping the moving object, the informal requirements are as follows: **1.** When an undesirable event occurs the fingers need to be returned to their initial positions; **2.** To guarantee safety of tactile sensors and object impedance we require that the contact force not exceed a certain upper bound F_{max}. **3.** Total time for task execution should not exceed a certain time T_{max}.

3 Notation

We use basic notations and semantics used in duration calculus for requirements specification and verification. Standard operators $\vee, \wedge, \neg, \Leftarrow, \Leftrightarrow, \forall, \exists, ;$ ("chop") are used. Common abbreviations from duration calculus [19, 6] for state assertion Q, duration formulas $\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2$ and a positive time constant t are used (table 1), with rules of precedence:

first: $\neg, \Box, \Diamond$
second: $\vee, \wedge, ;$
third: $\Rightarrow, \rightarrow, \sim t \rightsquigarrow$

The logical operators are used for state assertions as well as duration formulas.

4 Control model

We use a hierarchical control structure where the top-level control system consists of the **Master controller** which supervises the overall task execution. Two **Finger controllers** supervise the task execution for each finger (figure 2). Our system's task is to detect, reach, and grasp a moving object, then lift and move it to some specific location autonomously, while relying only on sensor information. The task will be completed when the object is dropped in a certain location and fingers are returned to their initial positions. All these phases correspond to the **Master controller** states and from each state the **Finger controllers** are called.

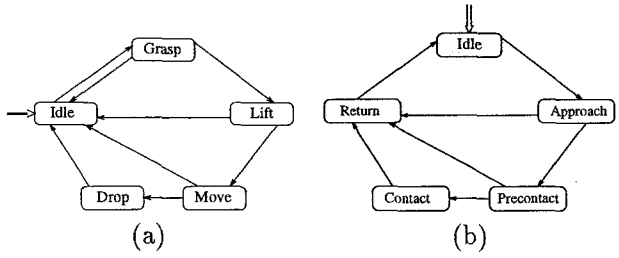

Figure 2: (a) Master controller automaton with states corresponding to the phases of the task. (b) The structure of the (left or right) finger controller.

4.1 Master controller

We define possible phase transitions by means of a finite state automaton with the set of states Q: {idle, grasp, lift, move, drop}. The corresponding phases, {Idle, Grasp, Lift, Move, Drop}, are defined when Q takes on the appropriate value, e.g. Idle $\hat{=}$ (Q = idle). The phase transition automaton shown graphically in figure 2 formally can be restricted to progress from phase to phase by progress constraints:

$$Phases \hat{=} Init \wedge Trans$$

where *Init* expresses that the automaton starts from an initial point in the Idle phase, $Init \hat{=} \lceil\ \rceil \rightarrow \lceil \mathtt{Idle} \rceil$, and *Trans* defines the phase transitions

$$\begin{aligned} Trans \ \hat{=}\ & \Box\ (\ (\lceil \mathtt{Idle} \rceil \rightarrow \lceil \mathtt{Grasp} \rceil) \\ & \wedge\ (\lceil \mathtt{Grasp} \rceil \rightarrow (\lceil \mathtt{Lift} \rceil\ \vee\ \lceil \mathtt{Idle} \rceil)) \\ & \wedge\ (\lceil \mathtt{Lift} \rceil \rightarrow (\lceil \mathtt{Move} \rceil\ \vee\ \lceil \mathtt{Idle} \rceil)) \\ & \wedge\ (\lceil \mathtt{Move} \rceil \rightarrow (\lceil \mathtt{Drop} \rceil\ \vee\ \lceil \mathtt{Idle} \rceil)) \\ & \wedge\ (\lceil \mathtt{Drop} \rceil \rightarrow \lceil \mathtt{Idle} \rceil)\ ). \end{aligned}$$

Predicate *PhaseReq* monitors each phase requirements .

$$\begin{aligned} PhaseReq \ \hat{=}\ & \Box\ (\ IdleReq \wedge GraspReq \wedge\ LiftReq \\ & \wedge MoveReq \wedge\ DropReq\ ) \end{aligned}$$

Due to space limitations we specify only *IdleReq* and *GraspReq*. In each of the following formulas ε denotes an upper bound on the duration of the phase and δ denotes an upper bound on the duration of the transition from one phase to another.

It is required that a transition from the Idle phase occurs within δ_I, after object information is obtained.

$$IdleReq \ \hat{=}\ (\lceil \mathtt{Idle} \rceil \wedge \lceil ObjInf \rceil \Rightarrow \ell \leq \delta_I\ )$$

The goal of the phase **Grasp** is to achieve a stable contact with the object. The phase **Grasp** enters phase **Lift** within δ_G, only if both fingers achieve stable contact.

$$bothContact \triangleq leftCont \wedge rightCont$$

If either left or right finger fails to accomplish a subtask in any of the sub-phases of **Grasp** , then Finger controller switches to the **leftRet** or **rightRet** respectively and the transition function, *Fail*,

$$Fail \triangleq \Diamond(\lceil \texttt{leftRet} \rceil \vee \lceil \texttt{rightRet} \rceil)$$

switches control of the Master controller to the phase **Idle**. Phase **Grasp** is stable for at least δ_G and lasts at most ε_G beyond *Fail* ($\delta_G \leq \varepsilon_G$).

$$\begin{aligned} &GraspReq \hat{=} \\ &\Box \quad ((leftReq \wedge leftTrans \wedge rightReq \wedge rightTrans) \\ &\wedge \quad (\lceil \texttt{Grasp} \rceil \wedge \lceil bothContact \rceil \Rightarrow \textstyle\prod \sim \delta_G \rightsquigarrow \lceil \texttt{Lift} \rceil) \\ &\wedge \quad (\lceil Fail \rceil \wedge \ell \leq \delta_G \Rightarrow \neg\Diamond(\lceil \texttt{Grasp} \rceil; \lceil \neg \texttt{Grasp} \rceil)) \\ &\wedge \quad (\lceil Fail \rceil \Rightarrow \textstyle\prod \sim \varepsilon_G \rightsquigarrow \lceil \texttt{Idle} \rceil)) \end{aligned}$$

The predicate *GraspReq*, a conjunction of four predicate formulas, evaluates to the *true* value only if each of the formulas has value *true*.

4.2 Finger controller

This subsection describes the lower level of control. The **Finger controllers** are responsible for supervision of task execution for both fingers. While in the phase **Grasp** and object geometry information is received, the **Master controller** commands the **Finger controllers** to execute their tasks - contact the object. While approaching the object each finger goes through the same phases, we limit our presentation to the left finger. The problem of synchronization of two fingers operating simultaneously will be discussed next.

Boolean functions will be used to formalize the task:

Fdist, leftCont, INside.

Fdist tells us whether there exist a sufficient distance between two fingertips:

$$Fdist = \begin{cases} 1, & \text{if } (x_{right} - x_{left}) > \sigma \\ 0, & \text{otherwise} \end{cases}$$

where x_{right} and x_{left} are x-coordinates of the right and left fingertips, respectively. Because of the coordinate system used (see figure 1) only x-components are needed to insure that fingertips are not overlapped. Each finger goes through the phases:

Idle: Entered from top-level phase **Grasp** when (*ObjInf* = 1); transition before *ObjInf* has lasted δ_i.

Approach: Reach the boundary of uncertainty region of object position as fast as possible, monitoring fingertip coordinates and velocity; enters **PreCont** phase if (*INside* = 1).

PreContact: Monitors force, velocity, and distance between left and right fingers; enters the **Cont** phase if contact is sensed and (*Fdist* = 1).

Contact: Maintains contact, adjusting force level, F_c.

Return: Returns finger to the initial position and then enters **Idle**.

For phases **Appr, PreCont, Cont** there is an error recovery procedure of returning to **Idle** through the phase **Ret**, so that, whenever a failure of any of the phases has occurred, both fingers go to the initial position and the system to **Idle**.

Phase transitions are defined by the automaton shown in Figure 2 and predicate

$$\begin{aligned} leftTrans \hat{=} \ &\Box((\textstyle\prod \rightarrow \lceil \texttt{Idle} \rceil) \wedge (\lceil \texttt{Idle} \rceil \rightarrow \lceil \texttt{Appr} \rceil) \\ &\wedge \ (\lceil \texttt{Appr} \rceil \rightarrow (\lceil \texttt{PreCont} \rceil \vee \lceil \texttt{Ret} \rceil)) \\ &\wedge \ (\lceil \texttt{PreCont} \rceil \rightarrow (\lceil \texttt{Cont} \rceil \vee \lceil \texttt{Ret} \rceil)) \\ &\wedge \ (\lceil \texttt{Cont} \rceil \rightarrow \lceil \texttt{Ret} \rceil)) \end{aligned}$$

The requirements for the left finger are

$$\begin{aligned} leftReq \hat{=} \Box(&leftIdleReq \wedge leftApprReq \wedge leftPrecontReq \wedge \\ &leftContReq \wedge leftRetReq) \end{aligned}$$

and we can specify each phase requirements. The **Idle** phase for the left finger (**leftIdle**) is left within time interval δ_i if the information about the object has arrived (*ObjInf* = 1).

$$leftIdleReq \triangleq (\lceil \texttt{leftIdle} \rceil \wedge \lceil ObjInf \rceil \Rightarrow \ell \leq \delta_i)$$

The **leftAppr** phase lasts at most ε_a. This phase switches to the **leftPreCont** phase within δ_a if the finger enters the object domain (*INside* = 1). The formal requirements for the **leftAppr** phase are

$$\begin{aligned} &leftApprReq \hat{=} (\lceil \texttt{leftAppr} \rceil \Rightarrow \ell \leq \varepsilon_a) \\ &\wedge(\lceil \texttt{leftAppr} \rceil \wedge \lceil INside \rceil \Rightarrow \textstyle\prod \sim \delta_a \rightsquigarrow \lceil \texttt{leftPreCont} \rceil) \\ &\wedge(\lceil Fail \rceil \Rightarrow \textstyle\prod \sim \delta_a \rightsquigarrow \lceil \texttt{leftRet} \rceil) \end{aligned}$$

The duration of the phase **leftPreCont** is at most ε_p. The phase **leftCont** is entered within δ_p, if event *leftCont* is detected and a certain distance σ between the fingertips exists. The phase enters **leftRet** within δ_p on $\neg Fdist$ or *Fail* .

$$\begin{aligned} &leftPrecontReq \hat{=}(\lceil \texttt{leftPreCont} \rceil \Rightarrow \ell \leq \varepsilon_p) \\ &\wedge(\lceil leftCont \rceil \wedge \lceil Fdist \rceil \Rightarrow \textstyle\prod \sim \delta_p \rightsquigarrow \lceil \texttt{leftCont} \rceil) \\ &\wedge(\lceil \texttt{leftPreCont} \wedge \neg Fdist \rceil \Rightarrow \textstyle\prod \sim \delta_p \rightsquigarrow \lceil \texttt{leftRet} \rceil) \\ &\wedge(\lceil Fail \rceil \Rightarrow \textstyle\prod \sim \delta_p \rightsquigarrow \lceil \texttt{leftRet} \rceil) \end{aligned}$$

The phase **leftCont** lasts at most ε_c, maintains the force level and is allowed to lose *Contact* for not longer than δ_{cont}. The phase **leftCont** enters the **leftLift** within time δ_c if function *bothContact* evaluates to 1 and enters **leftRet** if *Fail* = 1.

$$\begin{aligned} leftContReq \ &\hat{=} (\lceil \texttt{leftCont} \rceil \Rightarrow \ell \leq \varepsilon_c) \\ &\wedge \ (\lceil \neg leftCont \rceil \Rightarrow \ell \leq \delta_{cont}) \\ &\wedge \ (\lceil bothContact \rceil \Rightarrow \textstyle\prod \sim \delta_c \rightsquigarrow \lceil \texttt{leftLift} \rceil) \\ &\wedge \ (\lceil Fail \rceil \Rightarrow \textstyle\prod \sim \delta_c \rightsquigarrow \lceil \texttt{leftRet} \rceil) \end{aligned}$$

$$1)\,GraspReq \wedge \lceil \texttt{leftIdle} \wedge \texttt{rightRet} \wedge Fail \rceil \wedge \delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil \;\Rightarrow\; (\ell \le \delta_i + \delta_a + \delta_r ; \lceil \texttt{leftIdle} \wedge \texttt{rightIdle} \rceil) \wedge (\delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil)$$
$$2)\,GraspReq \wedge \lceil \texttt{leftAppr} \wedge \texttt{rightRet} \wedge Fail \rceil \wedge \delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil \;\Rightarrow\; (\ell \le \delta_a + \delta_r ; \lceil \texttt{leftIdle} \wedge \texttt{rightIdle} \rceil) \wedge (\delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil)$$
$$3)\,GraspReq \wedge \lceil \texttt{leftPreCont} \wedge \texttt{rightRet} \wedge Fail \rceil \wedge \delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil \;\Rightarrow\; (\ell \le \delta_p + \delta_r ; \lceil \texttt{leftIdle} \wedge \texttt{rightIdle} \rceil) \wedge (\delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil)$$
$$4)\,GraspReq \wedge \lceil \texttt{leftCont} \wedge \texttt{rightRet} \wedge Fail \rceil \wedge \delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil \;\Rightarrow\; (\ell \le \delta_c + \delta_r ; \lceil \texttt{leftIdle} \wedge \texttt{rightIdle} \rceil) \wedge (\delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil)$$
$$5)\,GraspReq \wedge \lceil \texttt{leftRet} \wedge \texttt{rightRet} \wedge Fail \rceil \wedge \delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil \;\Rightarrow\; (\ell \le \delta_r ; \lceil \texttt{leftIdle} \wedge \texttt{rightIdle} \rceil) \wedge (\delta_G \le \ell \le \varepsilon_G ; \lceil \texttt{Idle} \rceil)$$

Table 2: Cases for the grasp requirement predicate.

The `leftRet` phase is required to enter `leftIdle` after time δ_r and to return fingertip to the initial position within the same interval of time.

$$leftRetReq \;\hat{=}\; (\; \lceil \texttt{leftRet} \rceil \Rightarrow \ell \le \delta_r \;) \wedge (\; \lceil \texttt{leftRet} \rceil \Rightarrow \textstyle\prod \sim \delta_r \rightsquigarrow leftInitPos \;)$$

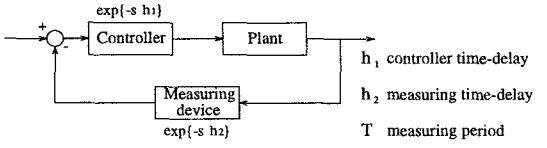

Figure 3: Close-loop control system with two time-delays.

5 Verification of the task requirements

Verification of the task requirements is a formal way to guarantee that a certain task will be solved, or equally, that the task requirements will be satisfied under given assumptions. This statement can be expressed by the predicate formula

$$Phases \wedge PhaseReq \Rightarrow \Box\; (\; Req_1 \wedge Req_2 \wedge Req_3 \;).$$

The total system requirements consist of three different requirements which are defined and verified below.

$1)\, Phases \wedge PhaseReq \Rightarrow Req_1 \qquad (initialization)$

$2)\, Phases \wedge PhaseReq \Rightarrow Req_2 \qquad (force-constraint)$

$3)\, Phases \wedge PhaseReq \Rightarrow Req_3 \qquad (time-constraint)$

Verification of Req_1 *(initialization)*: The first requirement may seem rather trivial, it states that whenever our "Master" is in the `Idle` state , each "Finger" has to be in the `Idle` state as well. Satisfying this requirement is analogous to sequentially operating systems in that each control cycle has to start from the same state. Formally this requirement can be written as

$Req_1 \;\hat{=}\; \lceil \texttt{Idle} \rceil \Rightarrow \lceil \texttt{leftIdle} \wedge \texttt{rightIdle} \rceil$

One should consider different cases depending on which state preceded the `Idle`.

$$\lceil \neg \texttt{Idle} \rceil ; \lceil \texttt{Idle} \rceil \Rightarrow \textstyle\prod ; \lceil \texttt{Idle} \rceil \;\vee\; \lceil \texttt{Grasp} \rceil ; \lceil \texttt{Idle} \rceil \vee \lceil \texttt{Lift} \rceil ; \lceil \texttt{Idle} \rceil \vee \lceil \texttt{Move} \rceil ; \lceil \texttt{Idle} \rceil \vee \lceil \texttt{Drop} \rceil ; \lceil \texttt{Idle} \rceil$$

The first case is trivial. Since there are no states preceding the `Idle` state, initial positions of the fingertips should be the same. The second case, from *GraspReq*, says that the phase `Idle` can be entered from phase `Grasp` only if the function *Fail* is satisfied.

$$\begin{aligned} & GraspReq \;\wedge\; (\lceil \texttt{Grasp} \rceil ; \lceil \texttt{Idle} \rceil) \\ & \Rightarrow true ; \lceil \texttt{Grasp} \wedge Fail \wedge \delta_G \le \ell \le \varepsilon_G \rceil ; \lceil \texttt{Idle} \rceil \\ & \Rightarrow true ; \lceil \texttt{Grasp} \wedge (\texttt{leftRet} \vee \texttt{rightRet}) \\ & \quad \wedge\, \delta_G \le \ell \le \varepsilon_G \rceil ; \lceil \texttt{Idle} \rceil \end{aligned}$$

Without loss of generality we suppose that the right finger is in the phase `rightRet`. Depending on the left finger phase the cases are given in table 2. Because there are no lower bounds on transitions from phase to phase, the case 1) is possible, and then it takes at most $\delta_i + \delta_a + \delta_r$ time to reach `leftIdle`. A case by case analysis shows that Req_1 will be satisfied as long as $\delta_G \ge \max(\delta_i + \delta_a, \delta_p, \delta_c) + \delta_r$.

Verification of Req_2 *(force constraint)*: In verifying Req_2 we guarantee that the force level of the finger-object contact force during the phase `Grasp` will never exceed some upper bound (F_{max}). This requirement is supposed to provide safe and robust operation for the force and/or tactile sensors.

To verify Req_2, the sub-phases `PreCont` and `Cont` are considered. These two sub-phases are safety critical; the force level could possibly exceed the allowed upper bound in one of these phases.

We make two assumptions about the hardware involved in the experiment: 1) The production line moves with a velocity v_{obj} which can fluctuate within an interval $v_{obj} \in [v_{obj}^-, v_{obj}^+]$ and 2) The algorithm used for image processing guarantees the accuracy of the image within some known margin. Both of this assumptions can be embedded to describe the growing boundary of object position uncertainty as a function of time.

When the fingertip enters the object uncertainty region the boolean function *INside* switches control mode from phase `Approach` to phase `PreCont`. In phase `PreCont` we start monitoring force level. To guarantee that the contact force level never exceeds its upper bound we consider the worst-case situation, and find parameters to satisfy:

$$Req_2 \;\hat{=}\; \forall t (F(t) \le F_{max}).$$

To do this, we model the "worst case" contact. Let the period of the control cycle "measurement - control - plant" be equal to T. We consider two types of delays - controller time-delay h_1 and delays in measurements h_2 (see figure 3). *A priori*, by testing the finger

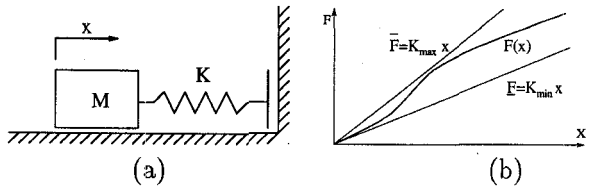

Figure 4: Worst case impact model and contact force approximation.

on its resistance to the contact, data of the contacting force as a function of displacement $F(x)$ can be obtained [8]. This function can be upper- and lower- bound approximated by the linear functions $\overline{F}(x) = K_{max}x$ and $\underline{F}(x) = K_{min}x$, respectively (see figure 4).

The next step in the verification is to model the dynamical system. In order to make a reliable prediction about the system behavior a good model of the dynamic system is needed. The model used has to be at least an "upper bound model" which guarantees that the real physical system will have contact force less than that predicted by the system.

In our worst case analysis we model the interaction between soft finger tip and environment as a mass-spring system with no energy dissipation (no damper in the model). The elastic force modeled as a spring is the upper bound of the contact force. The last assumption follows from the estimations on the contact force (see figure 4 (b)) $K_{min}\Delta x \leq F(x) \leq K_{max}\Delta x$.

By the controller design, the fingers approach the object with constant velocity v_0 (relative to the object). In the phase **PreCont** the control law does not change (the system is attempting to keep the same velocity v_0). However, in this phase the force sensors are turned on - the system is prepared to switch to the phase **Contact**. The force is measured every T seconds. The event contact is detected if the force measurement we receive is greater or equal to some threshold F_c.

The worst-case scenario is illustrated in figure 5. Assume that the force was measured at time t_0- just before the sensed force reaches the value F_c. Since $F < F_c$ we do not switch our control to the next mode, but continue motion with a control law U_1 designed for constant velocity tracking.

The next sampling occurs after period T. Force has now exceeded F_c. During the time $(h_1 + h_2)$ the event contact is recognized and a new control function U_2 is calculated and applied. Starting from $x(t_1)$ the system is trying to stabilize the contact force to the value F_c, that is, to maintain to the position $x = F_c/K_{max}$ (figure 5). Due to inertia the system displaces the finger to the position $x(t_2)$. The maximum force occurs at time t_2(figure 5) and in worst case will be equal to

$$\bar{F} = F_c + K_{max}\Delta x_1 + K_{max}\Delta x_2 \qquad (1)$$

where $\Delta x_1 = x(t_1) - x(t_0)$ and $\Delta x_2 = x(t_2) - x(t_1)$.

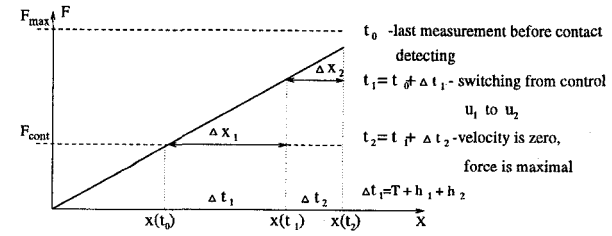

Figure 5: Maximum force estimation accounting delays, sampling period, and inertia of the control dynamic system.

Depending on the control function U_i $(i = 1, 2)$, our control dynamic system can be described by the system of differential equations

$$\begin{aligned} M\ddot{x} &= U_1 - K_{max}x, \qquad \text{for } t \in [0, t_1] \\ M\ddot{x} &= U_2 - K_{max}x, \qquad \text{for } t \in (t_1, t_2] \end{aligned} \qquad (2)$$

where the system can be stabilized by a PD-controller

$U_i = K_p^i(x_d^i - x) + K_d^i(\dot{x}_d^i - \dot{x})$ $(i = 1, 2)$.

Gains K_p^i and K_d^i are chosen to stabilize to the desired displacement and velocity, x_d^i and $\dot{x}_d^i$. Using U_i in (2) obtains the general form of the governing equations

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = At + B \qquad (3)$$

where

$$\left.\begin{array}{ll} 2\zeta\omega_n = K_d^1/M\ ; & \omega_n^2 = (K_{max}+K_p^1)/M \\ A = K_p^1 v_0/M\ ; & B = K_d^1 v_0/M \end{array}\right\} \text{for } t \in [0, t_1]$$

$$\left.\begin{array}{ll} 2\zeta\omega_n = K_d^2/M\ ; & \omega_n^2 = (K_{max}+K_p^2)/M \\ A = 0\ ; & B = K_p^2 F_c/MK_{max} \end{array}\right\} \text{for } t \in (t_1, t_2] \qquad (4)$$

The complete solution of (3) is (see e.g. [11])

$$x = \mathrm{e}^{-\zeta\omega_n t}(C\cos\omega_d t + D\sin\omega_d t) + \frac{At+B}{\omega_n^2} - \frac{2A\zeta}{\omega_n^3} \qquad (5)$$

where $\omega_d = \omega_n\sqrt{1-\zeta^2}$. Coefficients C and D can be found from the initial conditions for each interval

$$\left.\begin{array}{l} x|_{t=0} = 0 \\ \dot{x}|_{t=0} = v_0 \end{array}\right\} t \in [0, t_1] \qquad \left.\begin{array}{l} x|_{t=0} = 0 \\ \dot{x}|_{t=0} = \dot{x}(t_1) \end{array}\right\} t \in (t_1, t_2]$$

Δx_1 can be found by substituting $\Delta t_1 = T + h_1 + h_2$ as the final time for the first stage in the general solution (5). Δx_2 can be found in two steps: first, find the time to obtain the zero velocity in $x(t_2)$ (by differentiating (5) and substituting for zero velocity) and second, substitute the obtained time Δt_2 into (5) with the constants and initial condition corresponding to the second case. Using Δx_1 and Δx_2 in (1), we can determine the maximum possible contact force which may occur due to time delays h_1, h_2, sampling period T, and the system inertia. We will get the expression of $\bar{F}$ as a function of system parameters:

$$M, v_0, F_c, K_{max}, K_p^1, K_d^1, K_p^2, K_d^2, T, h_1, h_2$$

Now our verification consists of answering on the question of whether the parameters of the system can be

chosen such that the inequality $\bar{F} \leq F_{max}$ holds. We are *not* concentrating here on the problem of stability: it is assumed the the gains K_p^1, K_d^1, K_p^2, and K_d^2 can be determined, taking into account the effect of the time-delays and sampling period [13].

Verification of Req_3: To verify Req_3 we must prove that the execution time of each phase is bounded and that the sum of the time bounds is less then T_{max}, i.e. $T_{max} \geq T_G + T_L + T_M + T_D$. Here we prove only that the `Grasp` phase is bounded. `Grasp` is bounded, $\lceil \texttt{Grasp} \rceil \Rightarrow \ell \leq T_G$ follows because each of the sub-phases are bounded:

$$
\begin{aligned}
& \mathit{Phases} \wedge \mathit{GraspReq} \wedge (\lceil \texttt{Grasp} \rceil; \lceil \neg \texttt{Grasp} \rceil) \\
\Rightarrow\ & \lceil \texttt{Grasp} \rceil; \lceil \texttt{Idle} \rceil \vee \lceil \texttt{Grasp} \rceil; \lceil \texttt{Lift} \rceil \\
\Rightarrow\ & \mathit{true}; \lceil \texttt{Grasp} \wedge \mathit{Fail} \rceil; \lceil \texttt{Idle} \rceil \vee \\
& \mathit{true}; \lceil \texttt{Grasp} \wedge \mathit{bothContact} \rceil; \lceil \texttt{Lift} \rceil \\
\Rightarrow\ & (\ell \leq \texttt{t} + \varepsilon_G); \lceil \texttt{Idle} \rceil \vee (\ell \leq \texttt{t} + \delta_G); \lceil \texttt{Lift} \rceil
\end{aligned}
$$

We find `t` as a maximum possible interval between the beginning of `Grasp` and the occurrence of *Fail* or *bothContact*. The `Grasp` phase is entered on *ObjInf* (*ObjInf* = 1). From the *leftIdleReq* and *rightIdleReq* follows that δ_i is the maximum time spend in the `leftIdle` or `rightIdle` sub-phases. All other sub-phases are also time bounded and $\delta_G \leq \varepsilon_G$ thus we can deduce

$$
\begin{aligned}
& (\texttt{t} \leq \delta_i + \varepsilon_a + \varepsilon_p + \varepsilon_c) \\
& \Rightarrow (\lceil \texttt{Grasp} \rceil \Rightarrow \ell \leq \delta_i + \varepsilon_a + \varepsilon_p + \varepsilon_c + \varepsilon_G)
\end{aligned}
$$

The verification proofs for the remaining phases are analogous to the one presented here.

6 Discussion.

We have presented a methodology for a system designer to specify and verify overall system behavior. Although the evaluation of logical formulas may seem unusual to a control systems engineer, this level of analysis enables the designer to specify and reason about real-time and logical constraints in dynamical systems without the explicit mention of time instants. We hope in the future to be able to automate this verification process, a necessary step as the verification process becomes tedious as the complexity of the system increases. Eventually, the developed techniques will result in a more efficient, systematic, and reliable design process.

References

[1] H. Allen. Towards a general theory of action and time. *Artificial Intell.*, 23:123–154, 1984.

[2] R. Alur, *et. al.* Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, 209–229, 1993.

[3] D. Auslander, *et al.* A Design and Implementation Methodology for Real-Time Control of Mechanical Systems, *Mechatronics*, 5(7), 1995.

[4] B. Berthomieu & M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. on Software Eng.*, 259–273, 1991.

[5] R. Brockett. Hybrid models for motion control systems. *Essays in Control*, 29–53, 1993.

[6] Z. Chaochen, C. Hoare, & A. Ravn. A calculus of durations. *Info. Proc. Lett.*, 40(5):269–276, 1991.

[7] Z.Chaochen, *et al.* Extended duration calculus for hybrid real-time systems. *Hybrid Systems*, 1993.

[8] M. Cutkosky. Manipulation control with dynamic tactile sensing. In *6th ISRR*, Hidden Valley PA, 1993.

[9] C. Fagerer, D. Dickmanns, & E. Dickmanns. Visual grasping with long delay time of a free floating object in orbit. *J. of Oceanic Eng.*, 1:53–68, 1994.

[10] B. Hove & J-J. Slotine. Experiments in robotic catching. *Proc. Amer. Control Conf.*, 1, 1991.

[11] M. James, *et al. Vibration of Mechanical and Structural Systems.* Harper & Row, 1989.

[12] M. Lemmon & P. Antsaklis. Inductively inferring valid logical models of continuous-state dynamical systems. *Theor. Comp. Sc.*, 138:201–210, 1995.

[13] M. Malek-Zavarei & M. Jamshidi. *Time-Delay Systems: analysis, optimization, and applications.* Elsevier Science Pub. Co., 1987.

[14] Z. Manna & A. Pnueli. Specification and verification of concurrent programs by ∀-automata. *Proc. ACM Symp. on Princ. Prog. Lang.*, pp 1–12, 1987.

[15] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. IEEE *Trans. Comput.*, C18(2):10-19, 1985.

[16] A. Nerode & W. Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. *Hybrid Systems*, 317–356, 1993.

[17] G. Niemeyer & J-J. Slotine. Stable adaptive teleoperation. *J. Oceanic Eng.*, 16(1):152–162, 1991.

[18] P. Ramadge & W. Wonham. The control of discrete event systems. In *Proc. IEEE*, Jan. 1993.

[19] A. Ravn, *et al.* Specifying and verifying requirements of real-time systems. IEEE *Trans. on Soft. Eng.*, 19(1), 1993.

[20] A. Ravn, *et. al.* Hybrid control of a robot - a case study. *Hybrid Systems*, 999:391–404, 1995.

[21] J. Skakkebaek, *et. al.* Specification of embedded real-time systems. *Proc. 4th Euromicro Workshop on Real-Time Systems*, pages 116–121, June 1992.

[22] L. Tavernini. Differential automata and their discrete simulators. *Nonlinear Analysis, Theory, Methods, and Applications*, 11:665–683, 1987.

[23] Y. Zhang & A. Mackworth. Constraint nets: a semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138:211–239, 1995.