

A Timing Model for Vision-Based Control of Industrial Robot Manipulators

Yanfei Liu, Adam Hoover and Ian Walker
Electrical and Computer Engineering Department
Clemson University
Clemson, SC 29634-0915
{lyanfei, ahoover, iwalker} @clemson.edu

Abstract

Visual sensing for robotics has been around for decades, but our understanding of a timing model remains crude. By timing model, we refer to the delays (processing lag and motion lag) between “reality” (when a part is sensed), through data processing (the processing of image data to determine part position and orientation), through control (the computation and initiation of robot motion), through “arrival” (when the robot reaches “reality”). In this work we introduce a timing model where sensing and control operate asynchronously. We apply this model to a robotic workcell consisting of a Stäubli RX-130 industrial robot manipulator, a network of six cameras for sensing, and an off-the-shelf Adept MV-19 controller. We demonstrate some experiments to show how the model can be applied.

Key words: workcell, timing model, visual servoing

Please send all correspondence to:

Yanfei Liu
Electrical & Computer Engineering Department
Clemson University
Clemson, SC 29634-0915
lyanfei@clemson.edu
voice: 864-656-7956
fax: 864-656-7220

Submitted as a regular paper on July 2003 to the IEEE Trans. on Robotics & Automation.

A Timing Model for Vision-Based Control of Industrial Robot Manipulators

Yanfei Liu, Adam Hoover and Ian Walker
 Electrical and Computer Engineering Department
 Clemson University
 Clemson, SC 29634-0915
 {lyanfei, ahoover, iwalker} @clemson.edu

Abstract—Visual sensing for robotics has been around for decades, but our understanding of a timing model remains crude. By timing model, we refer to the delays (processing lag and motion lag) between “reality” (when a part is sensed), through data processing (the processing of image data to determine part position and orientation), through control (the computation and initiation of robot motion), through “arrival” (when the robot reaches “reality”). In this work we introduce a timing model where sensing and control operate asynchronously. We apply this model to a robotic workcell consisting of a Stäubli RX-130 industrial robot manipulator, a network of six cameras for sensing, and an off-the-shelf Adept MV-19 controller. We demonstrate some experiments to show how the model can be applied.

Key words: workcell, timing model, visual servoing

I. INTRODUCTION

Figure 1 shows the classic structure for a visual servoing system [1]. In this structure, a camera is used in the feedback loop. It provides feedback on the *actual* position of something being controlled, for example a robot. This structure can be applied to a variety of systems, including eye-in-hand systems, part-in-hand systems and mobile robot systems.

In an eye-in-hand system [2], [3], [4], [5], [6], the camera is mounted on the end-effector of a robot and the control is adjusted to obtain the desired appearance of an object or feature in the camera. Gangloff [2] developed a visual servoing system for a 6-DOF manipulator to follow a class of unknown but structured 3-D profiles. Papanikolopoulos [3] presented algorithms to allow an eye-in-hand manipulator to track unknown-shaped 3-D objects moving in 2-D space. The object’s trajectory was either a line or an arc. Hashimoto [4] proposed a visual feedback control strategy for an eye-in-hand manipulator to follow a circular trajectory. Corke [5], [6] presented a visual feedforward controller for an eye-in-hand manipulator to fixate on a ping-pong ball thrown across the system’s field of view.

In a part-in-hand system [7], the camera is fixed in a position to observe a part which is grasped by a robot. The robot is controlled to move the part to some desired position. For example, Stavitzky [7] built a system to let the robot align a metal part with another fixed part. Since the part is always grasped by the manipulator, we can also say that the part is something being controlled. Or in the other words, we can say that how the object appears in the camera is controlled.

In some mobile robot problems [8], the camera is mounted over an environment to sense the actual position of the mobile robot as feedback to the controller. For example, Kim [8] built a mobile robot system to play soccer. The camera is fixed over the field and acts as a feedback position sensor. Here, the camera observes something which is directly controlled.

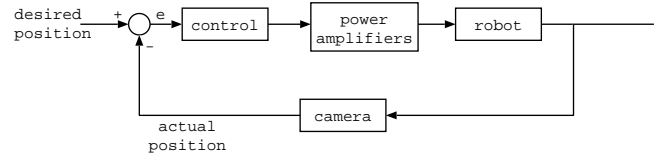


Fig. 1. Classical visual servoing structure.

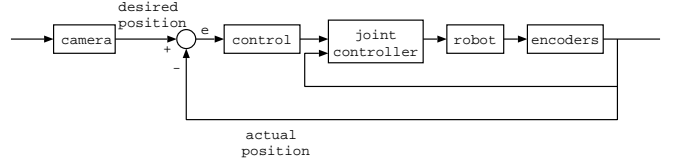


Fig. 2. Vision guided control structure.

All of these systems, regardless of where the camera is mounted, use the camera in the same control structure. In each case the system regulates how the object appears in the camera.

In this paper, we consider the problem where the camera is used to provide the *desired* or *reference* position to the internal robot controller. Figure 2 shows the structure for this system. There are several types of problems that fit this kind of system, where the object of interest cannot be controlled directly. For example, imagine a robot trying to pick up live chickens, or a robot trying to manipulate parts hanging on a swaying chain conveyor. Similar problems have been investigated in some works. Houshangi [9] developed a robot manipulator system to grasp a moving cylindrical object. The motion of the object is smooth and can be described by an auto-regressive (AR) model. Allen [10] demonstrated a PUMA-560 tracking and grasping a moving model train which moved around a circular railway. Miyazaki [11] built a ping pong robot to accomplish the ping pong task based on virtual targets. Nakai [12] developed a robot system to play volleyball with human beings. From the above systems, we notice that the motion of the object was limited to a known class of trajectories. In this paper, we seek to extend this to let the robot follow an unstructured (completely unknown) trajectory. This will be enabled in part by providing a generic timing model for this kind of system.

Our timing model considers the problem where image processing and control happen asynchronously. We are faced with the following three problems:

- 1) The maximum possible rate for complex visual sensing and processing is much slower than the minimum required rate for mechanical control.
- 2) The time required for visual processing introduces a significant lag between when reality is sensed and when the visual understanding of that reality (e.g. image tracking result) is available. We call this the *processing lag*.
- 3) The slow rate of update for visual feedback results in larger desired motions between updates, producing a lag in when the mechanical system completes the desired motion. We call this the *motion lag*.

Consider problem #1. A standard closed-loop control algo-

work	image processing rate (HZ)	control rate (HZ)	processing lag (ms)	motion lag (ms)
Papanikolopoulos et. al. [3]	10	300	100	–
Hashimoto et. al. [4]	4	1000	–	–
Corke and Good [5], [6]	50	70	48	–
Stavnitzy and Capson [7]	30	1000	–	–
Kim et. al. [8]	30	30	90	–
Houshangi [9]	5	36	196	–
Allen et. al. [10]	10	50	100	–
Miyazaki et. al. [11]	60	–	–	–
Nakai et. al. [12]	60	500	–	–
this work	23	250	151	130

TABLE I
SUMMARY OF RELATED WORK

rithm assumes that new data can be sensed on each iteration of control. Common industrial cameras operate at 30 Hz while common control algorithms can become unstable at rates less than several hundred Hz. Complex image processing tasks, such as segmentation, pose estimation, feature matching, etc., typically run even slower than 30 Hz, while control problems can require rates as high as 1K Hz. In general, this gap in rates will not be solved by the trend of increasing computational power (Moore’s Law). As this power increases, so will the amount of desired visual processing, and so will the complexity of the control problem. In this paper we propose to address this problem directly by modeling the visual sensing, processing, and control processes as having fundamentally different rates, where the sensing and processing are at least one order of magnitude slower than the control.

Problem #2 is a consequence of the complexity of the image processing operations. There is always a lag (processing lag) between reality and when the result from processing a measurement of that reality is available. In a high-speed (e.g. 1 KHz) closed-loop control, this lag can usually be ignored. But as the processing complexity increases, a non-negligible lag is introduced between when the image was acquired (reality) and when the image processing result is available (e.g. estimate of object position). We incorporate an estimate of the processing lag directly into our timing model.

Problem #3 is also a consequence of the slow rate of visual sensing and processing. In a high-speed closed-loop control, the motion executed between iterations is expected to be small enough to be completed during the iteration. The motion lag (time it takes to complete the motion) is considered negligible. But as the sensing rate slows, the tracked reality (e.g. object) moves farther between iterations, requiring the mechanical system (e.g. robot) to also move farther between iterations. As a consequence, it is possible to have a system that has not completed the desired set point motion prior to the next iteration of control. We address this problem by directly incorporating an estimate of the motion lag into our timing model.

Table I presents a summary of how previous works have featured and addressed these three problems. From this table we note that the first two of the three problems have been addressed to some extent in previous works. However, no work appears to have explicitly considered problem #3. All of these works neglect the motion time (motion lag) of the robot. One

work [10] noted this problem and used an $\alpha - \beta - \gamma$ predictor to compensate for it instead of explicitly modeling it. None of these works has considered the generic modeling of this type of system.

Some works synchronized the image processing rate and control frequency for a more traditional solution. In [2], the frequency of visual sensing, processing and control were all set to 50 HZ. Basically, the control frequency was synchronized to the image processing rate for simplicity. Simulation results of high frequency control, i.e. 500 HZ, were also shown in [2]. Performance of the high frequency controller was, as expected, better than the low frequency version motivating a more thorough investigation of a generic timing model to solve the problem. Corke [6] and Kim [8] presented timing diagrams to describe the time delay. Based on the timing diagrams, these works tried to use discrete time models to model the systems. In order to do this, the authors simplify these asynchronous systems to single-rate systems. It is well known that the discrete time model can only be applied into single-rate systems or systems where the control rate and the vision sensing rate are very close. However, from Table I, we notice that most real systems do not satisfy this condition. Therefore, in this paper we propose a continuous generic timing model to describe asynchronous vision-based control systems.

The remainder of this paper is organized as follows. In Section II, we describe our generic timing model, and then apply this model to an industrial robot testbed that uses a network of cameras to track objects in its workcell. In Section III, We demonstrate the importance of the application of our model by using it to derive a “lunge” expression that lets the robot intercept an object moving in an unknown trajectory. Finally, we conclude the paper in Section IV.

II. METHODS

Figure 3 illustrates our timing model. From top to bottom, each line shows a component of the system in process order (e.g. sensing comes before image processing). The horizontal axis represents time. We use this model to quantify the processing lag and motion lag of the system. The processing lag is the time between reality (e.g. the actual position of an object) and when an estimate of that reality is available (e.g. the tracking result from processing the image of reality). Similarly, the motion lag is the time between when the control

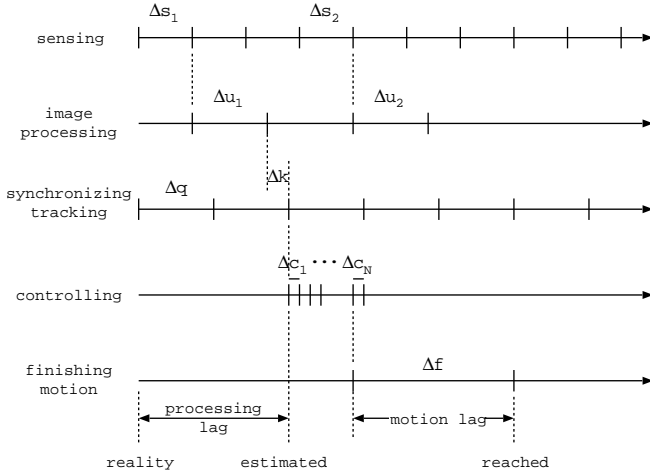


Fig. 3. Timing model for estimating the lag and latency.

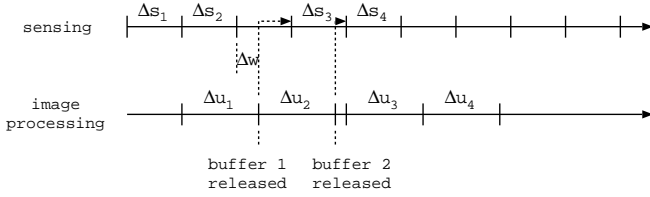


Fig. 4. Timing model using double buffering for processing image data.

command is issued and when the mechanical system finishes the motion.

The sensing and control processes operate at fixed intervals Δs and Δc , where $\Delta s > \Delta c$ (sensing is slower than control). The time required for all image processing and tracking operations is designated Δu . This processing starts when an input buffer is filled with image data (on a clock or sync signal defined by the sensing line). An input buffer cannot be filled with new image data until the processing of the previous image data in that buffer is completed. In Figure 3, this is why Δs_2 starts on the next sync after the end of Δu_1 .

Figure 3 depicts the case where $\Delta u > \Delta s$ (the processing takes longer than the sensing interval) and when there is only one image buffer. Figure 4 depicts the case where two input buffers are used (commonly called “double buffering”). In this case, a second image buffer is being filled while the image data in the first buffer is being processed. Double buffering increases the lag (note the extra time Δw between the end of Δs_2 and the start of Δu_2) but increases the throughput (note the greater number of images processed in Figure 4 as compared to Figure 3).

Figure 5 depicts the even more complicated case where double buffering happens consecutively. For example, a framegrabber can be equipped with enough memory to double buffer images on the framegrabber itself as they are digitized. This double buffer can then feed a second double buffer residing in main (host) memory. Although this again increases throughput, Figure 5 shows how it also increases the lag. Each box indicates the time spent by an image at each stage. The interval Δw_m is the time an image spends waiting in

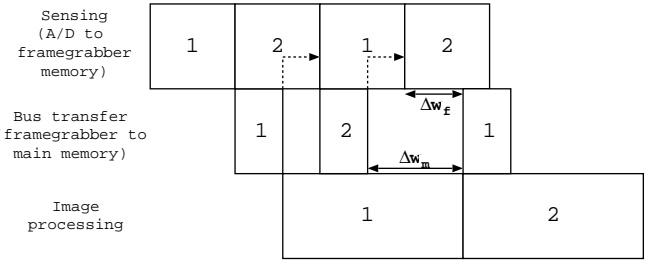


Fig. 5. Timing model of using consecutive double buffering.

host memory for the completion of processing of the previous image. This is similar to the term Δw in Figure 4. The interval Δw_f is the time an image spends waiting in the framegrabber for a host buffer to become available. In this case, the image on the framegrabber is waiting for the host to finish processing the previous image residing in the buffer needed for transfer of the new image.

In all of these cases, we have assumed that the processing takes longer than the sensing interval ($\Delta u > \Delta s$). In the case where $\Delta u < \Delta s$ (the processing is faster than the sensing rate), double buffering makes it possible to process every image. In any case, there is always a minimum lag of $\Delta s + \Delta u$, but depending on the buffering used and the relation of Δs to Δu the lag can be larger.

In order to handle all of these cases, we introduce a synchronous tracking process (line 3 in Figure 3) operating at a rate of Δq . The tracking line takes the most recent result from the image processing line and updates it for any additional delay (depicted as Δk) using a Kalman filter. In general, we desire $\Delta q \approx \Delta u$ so that tracking is updated approximately as fast as new results become available. A secondary benefit of the synchronous tracking line is that it satisfies standard control algorithm requirements that assume synchronous input data. Without this line, the results from image processing can arrive asynchronously (as in Figures 4 - 5).

The fourth and fifth lines in Figure 3 represent the control process and completion of motion. We consider the case where the distance traveled by an object between tracking updates is larger than a robot could safely or smoothly move during a single iteration of control. The control is therefore broken up into a series of N sub-motion commands occurring at a rate of Δc . Additionally, we expect the motion requested by any new iteration of control to take Δf time to complete. Figure 3 depicts the case where control commands are cumulative (each new control command is relative to the last commanded goal). In Section II-A we describe our prototype, which uses an off-the-shelf Adept controller that operates in this manner. For this controller, the motion is completed some time Δf after the last given control command. It is of course possible to have an open architecture controller that finishes the motion just prior to the next iteration of control. In this case, $\Delta f \approx \Delta c$.

Once values are known for the variables Δs , Δu , Δq , Δc and Δf , it is possible to derive various expressions for controlling a robot to solve specific problems, for example, to intercept a moving object. In the next section, we describe our prototype workcell and derivation of the timing variables.



Fig. 6. Our prototype dynamic workcell.

In Section III, we derive an expression for implementing a “lunge” of the robot to intercept an object moving with an a priori unknown trajectory.

A. Prototype

Figure 6 shows a picture of our prototype workcell for this project. We use a Stäubli RX130 manipulator with its conventional controller, the Adept Corporation model MV-19. In the V+ language (the software environment of the controller), we use the “Alter” command to implement real-time robot motion [13]. The “Alter” command can move the end-effector in an x, y, z direction, i.e. the “Alter” command works in Cartesian space. It can execute as quickly as every 2 ms. A network of six cameras surrounds the workcell, placed on a cube of aluminum framing. The cameras are wired to two imaging technology PC-RGB framegrabbers (A/D video converters) mounted in a Compaq Proliant 8500 computer. The Compaq has a standard SMP (system multiprocessor) architecture equipped with eight 550 MHz Intel Pentium 3-Xeon processors. In [14], we detailed the workcell configuration, calibration, image differencing and real-time robot motion planning. In [15], we presented some tracking experiments to show that our system can track different kinds of objects using continuous visual sensing.

Figure 7 shows a timing diagram describing the flow of information through the system, along with how long each step takes. Some of these estimates were derived analytically from knowledge of the hardware, while other estimates were derived from measurements taken while the system was operating. We will discuss each part in detail in the following paragraphs.

Figure 7 starts with an event that is happening in real time (e.g. an object moves). The cameras operate at 30 Hz using the standard NTSC (U.S. television) format, so that for this system $\Delta s = 33$ ms. The reality that is imaged takes 33 ms to transfer from the camera to the framegrabber (an NTSC signal uses its full sampling interval for transmission). At the receiving side of this wired transfer, the framegrabber is digitizing and buffering (on memory resident on the framegrabber) pixel values as they appear on the cable. We assume that the final pixel A/D takes negligible time, so that the digitized image

is stored on the framegrabber 33 ms after reality. The six cameras are synchronized on a common clock for the vertical sync (start of image). Each camera is greyscale, so that three cameras may be wired to the red, green and blue components of an RGB input on a framegrabber. Our system uses two framegrabbers wired in this manner. After 33 ms, all six images are digitized and residing in framegrabber memory.

The framegrabbers have microcontrollers that can operate as PCI bus masters, initiating transfers from framegrabber memory to main memory. The Compaq is programmed to use a ring buffer (with room to hold two sets of six images), to facilitate double buffering. While one set of six images is being processed, a second set can be transferred from framegrabber memory to main memory, at the same time. Assuming negligible traffic on the PCI bus, the time for this transfer can be computed as the total number of image bytes divided by the bandwidth of the bus: $(640 \times 480 \times 4 \times 2 \text{ bytes}) / (4 \text{ bytes} \times 33 \text{ MHz}) = 19$ ms, where the bandwidth is the theoretical maximum provided by the 32-bit 33 MHz PCI standard. Although each framegrabber is only capturing three channels of image data, four channels of bytes are transferred. The fourth channel is commonly called the alpha channel, and is empty for our purposes, but is used to keep image data on even word (32-bit) boundaries.

The image processing portion of our system creates a two-dimensional occupancy map of the space in a horizontal plane of interest in the workcell, locates the centroid of an object in this space, and displays the result on-screen [16]. Based upon empirical measurements of the run-time of the image processing methods on the Compaq, we observed them to take approximately 30 ms on average each iteration. This time can vary by approximately ± 2 ms depending upon the content of the images. We discuss the variances of our measurements in more detail at the end of this section. We also measured that the image display takes 14 ms on average. Therefore, the total processing time Δu for this system is $19 + 30 + 14 = 63$ ms.

The images may wait in buffers before being processed, due to our use of consecutive double buffering (see Figure 5). The waiting time can vary depending on the phasing of Δs and Δu as shown in Figure 7. The buffer in main memory is released right after the occupancy map is available. We noticed that after several iterations, the waiting time repeats in a pattern. The main memory waiting time Δw_m becomes a constant (39 ms) after 4 iterations. From Figure 7, we can observe that the PCI bus transfer always happens right after image processing finishes. Therefore, the main memory waiting time Δw_m equals the display time plus image processing time, minus PCI bus transfer time, i.e. $14 + 14 + 30 - 19 = 39$ ms. The framegrabber waiting time Δw_f is repeating in 3 numbers, 5 ms, 16 ms, 27 ms. So we take the average waiting time $\overline{\Delta w_f}$ as $(5 + 16 + 27) / 3 = 16$ ms. Thus we can get the total average waiting time $\overline{\Delta w} = 39 + 16 = 55$ ms. For our system, the syncing time Δk is a variable that we get in real-time. Adding up the appropriate terms $(\Delta s + \Delta u + \overline{\Delta w} + \Delta k)$, the processing lag for this system is $33 + 63 + 55 + \Delta k = (151 + \Delta k)$ ms. To unify the terms, we use Δl to express the computable partial lag time (151 ms).

The process that synchronizes the tracking result uses a

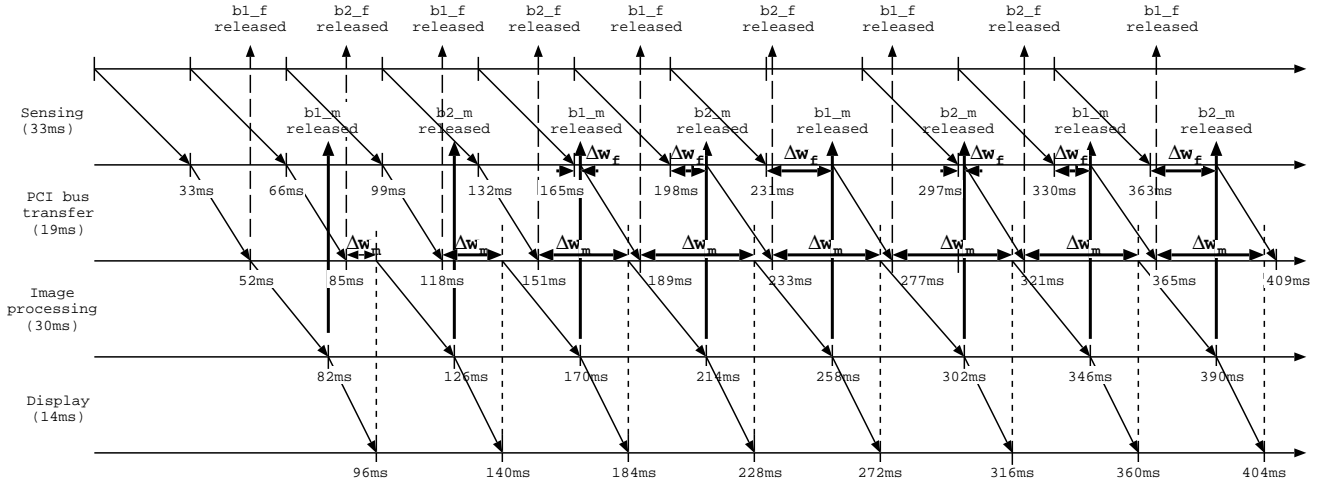


Fig. 7. Information flow through system.

Kalman filter to update the actual Δk for the given iteration. This position is sent through a 10 Mbit ethernet link from the Compaq to the Adept. Based on empirical measurements, these operations were observed to collectively take less than 1 ms. For this system we set $\Delta q = 40$ ms which is near but slightly under the occupancy map time plus the image display time (30 + 14 ms).

At this point the Adept (robot) has a new goal. This goal is directly forwarded to the motor-level controller through the “Alter” command available as part of the “Enhanced trajectory control” option of the V+ operating system [13]. According to the manufacturer (Adept), the maximum issue rate for the Alter command is 500 Hz (once every 2 ms), but through experimentation we observed that this rate could not always be maintained. Therefore we set $\Delta c = 4$ ms, issuing a new Alter every 4 ms. The precise details of the motor-level controller are proprietary to Adept Corporation and could not be determined. Therefore we determined Δf empirically through repeated measurements of the time it took to complete an Alter command. We varied the distance moved, direction moved, and initial condition (starting with the robot in motion at varying velocities, including at rest). Across all cases the time to complete an Alter was observed to vary from 120 to 150 ms, with a commonly occurring median value of 130 ms. We therefore set Δf to be 130 ms.

In our model we assume constants for most of the lag terms. It is important to note that all of these terms have variances, some of them having appreciable size in this context (more than 1 ms). For the sensing and processing terms, it would be ideal to timestamp each image upon acquisition and measure the terms precisely. However, in order to solve problems involving estimates into the future (for example to plan a catch of an object) it is necessary to have averages. Therefore we only note the variances here, and leave a more thorough understanding of their effect to future work.

In order to validate our estimate of total lag for our system, we conducted a series of experiments. Figure 8 shows a picture of the experimental setup. We set up another camera, completely external to our system, to observe its operation.

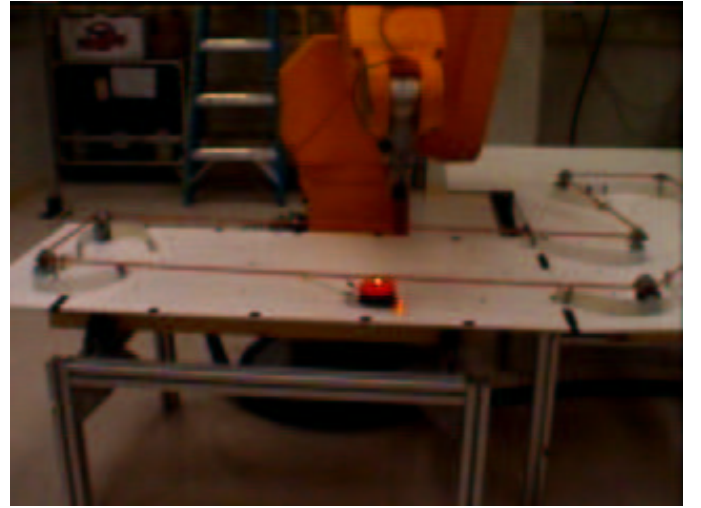


Fig. 8. Experimental setup for lag measurement.

A conveyor was constructed to move in a constant path at a constant velocity. A light bulb was attached to the conveyor and the system was directed to track the object, keeping the robot positioned above the light bulb. A small laser was mounted in the end effector of the robot, so that it pointed straight down onto the conveyor. The experiment was configured to make it possible for the external camera to estimate the distance between the light bulb and the footprint of the laser on the conveyor. Knowing the velocity of the light bulb, we were able to empirically measure the total lag of our system, and verify that it matched the estimate derived from our timing model. Complete details of this experiment can be found in a technical report [17].

III. EXPERIMENTS

In order to test our methods, we experiment with the problem of catching a moving object. Figure 9 depicts the scenario. The object is moving at an unknown constant velocity in a straight line. In this example, the object is moving in one dimension; however, we formulate the solution using vectors

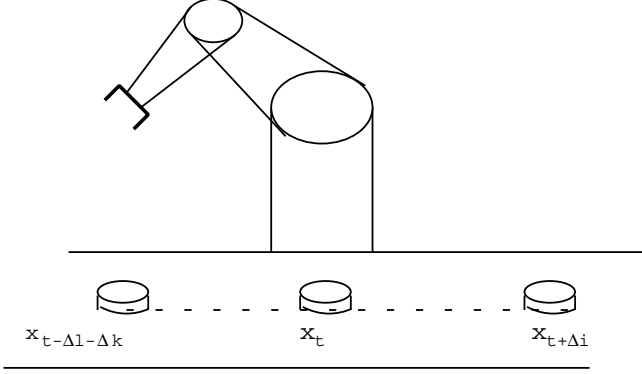


Fig. 9. Scenario for intercepting objects.

to indicate that the solution is also applicable to 2D and 3D problems. Due to the processing lag, the most recently measured position of the object is where the object was $(\Delta l + \Delta k)$ ms previously. We denote this location as $\vec{x}_{t-\Delta l-\Delta k}$. The current position of the object, i.e. the time when the robot starts to lunge towards the object, is denoted as \vec{x}_t . The current velocity of the object is denoted as \vec{v}_t , and is assumed to be equal to the last measured velocity $\vec{v}_{t-\Delta l-\Delta k}$. Therefore, the relationship between \vec{x}_t and $\vec{x}_{t-\Delta l-\Delta k}$ is described in the following equation:

$$\vec{x}_t = \vec{x}_{t-\Delta l-\Delta k} + \vec{v}_{t-\Delta l-\Delta k}(\Delta l + \Delta k) \quad (1)$$

It will take some amount of time Δi ms for the robot to reach the point of impact where it will catch the object. We denote the impact location as $\vec{x}_{t+\Delta i}$.

There is more than one way to consider implementing a catch. One approach is to keep the robot's position at the most recently measured position ($\vec{x}_{t-\Delta l-\Delta k}$) and then lunge to the final impact position. The advantage to this approach is that the robot is only moving to locations where the object has actually traveled. The disadvantage to this approach is that the robot must lunge a distance that covers the lag time plus the time to impact. A second approach is to keep the robot's position at the current position (\vec{x}_t), or even keep the robot's position near the predicted impact position ($\vec{x}_{t+\Delta i}$). If the object is moving at a constant velocity or in a fixed pattern, this approach will not suffer from misprediction and will always decrease the lunge distance. However, if the object motion is not so controlled, this approach could cause unfavorable behavior, for example increasing the lunge distance in the case where the object reverses direction. In either approach, we can describe the intercept problem as the following: if the robot desires to intercept the object at time t while following the object, how many control commands (N) should be issued between time t and the time when the robot intercepts the object, and what is the constant distance ($\Delta \vec{d}$) that each single control command should move.

For some problems, N is fixed. The solution for a fixed N problem involves only one equation:

$$\vec{x}_{t+\Delta i}[i] = \vec{x}_t[i] + \vec{v}_t[i](\Delta c \times (N - 1) + \Delta f) \quad (2)$$

For our problem, N is varying. Suppose that $\vec{x}_{t-\Delta q}$ is the

position where the robot was last commanded to move to, i.e. the position where the object is at time $t - \Delta q$, \vec{n} is the number of alters which will be executed, and d is the maximum distance a single alter can move. The solution now involves two equations:

$$\vec{x}_{t+\Delta i}[i] = \vec{x}_t[i] + \vec{v}_t[i](\Delta c \times (\vec{n}[i] - 1) + \Delta f) \quad (3)$$

$$|\vec{x}_{t+\Delta i}[i] - \vec{x}_{t-\Delta q}[i]| = \vec{n}[i] \times d \quad (4)$$

Combining Equations 3 and 4, based on the assumption that the object does not change velocity direction between $t \dots t - \Delta q$, we obtain:

$$\vec{n}[i] = \left\lceil \frac{(\Delta f - \Delta c) |\vec{v}_t[i]| + |\vec{x}_t[i] - \vec{x}_{t-\Delta q}[i]|}{d - \Delta c |\vec{v}_t[i]|} \right\rceil \quad (5)$$

When we solve for \vec{n} , there is a constraint:

$$|\vec{v}_t[i]| < \frac{d}{\Delta c} \quad (6)$$

Then N is chosen as the maximum element of vector \vec{n} . Therefore,

$$\Delta \vec{d}[i] = \frac{\vec{x}_{t+\Delta i}[i] - \vec{x}_{t-\Delta q}[i]}{N} \quad (7)$$

A. Experimental setup and results

To verify that our model is effective, we design an experiment to let our industrial manipulator catch a moving object. A small cylindrical object is dragged by a string tied to a belt moving at a constant velocity. The belt is moved by a DC motor. The object moves along a straight line. Figure 10 shows our experimental setup. The robot follows the object with the end-effector pointing straight down approximately 300 mm above the object. When the robot is commanded to intercept the object, the robot will lunge and cover the object on the table with a modified end-effector, a small plastic bowl. The diameter of the object is 70 mm, the diameter for the small bowl is 90 mm. Therefore, the error should be less than 10 mm on each side in order to successfully catch the object.

In order to test the applicability of our timing model, we conducted two sets of experiments. We set Δq to two different values, 40 and 80, in these two sets of experiments. We varied the voltage of the motor driving the conveyor to let the object move at three different velocities. For each velocity, we kept the voltage of the motor constant to make the object move in a relatively fixed velocity, and ran the experiment ten times. Table II shows the results. The velocity column is filled with the range of the average velocity in the ten experiments. The standard deviation column is the range of the standard deviation of the velocity of each experiment. The results indicate that if the object moves at a relatively fixed velocity, the robot catches the object 100% of the time independent of the velocity of the object and the position update time (Δq).

While fixed velocity experiments can verify the accuracy of our timing model, our true goal is to enable the system to catch objects moving in unstructured (completely unknown) trajectories, with non-negligible accelerations. Towards this

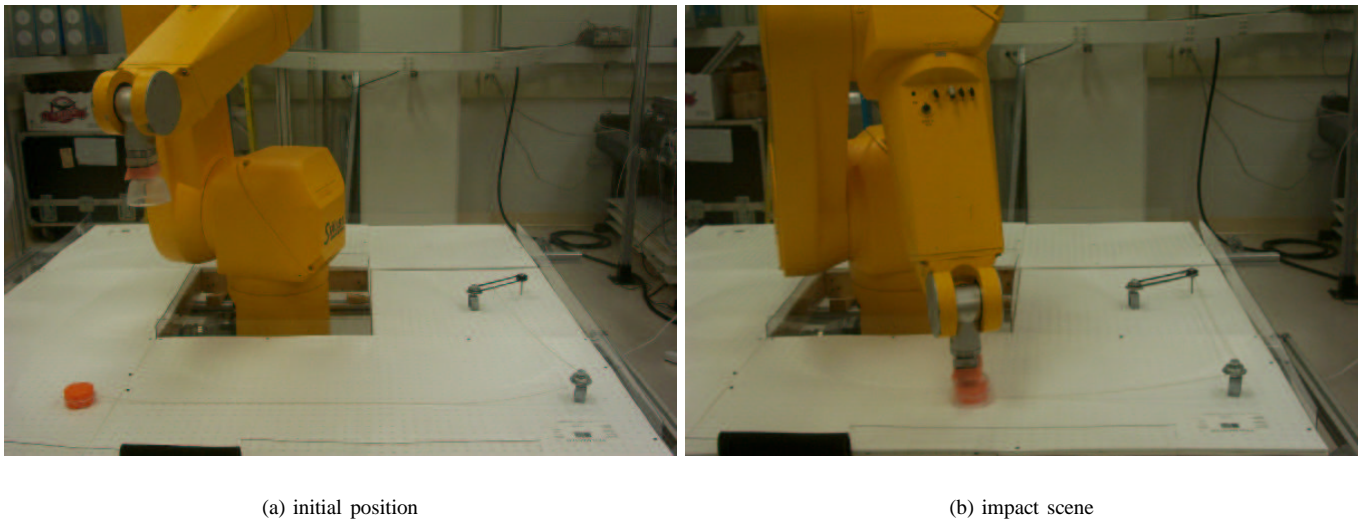


Fig. 10. Experimental setup for catching the moving object.

$\Delta q = 40$			$\Delta q = 80$		
velocity (mm/s)	stdev (mm/s)	catch percentage	velocity (mm/s)	stdev (mm/s)	catch percentage
84.4 – 97.4	1.3 – 3.8	100%	85.9 – 95.1	2.5 – 3.7	100%
129.8 – 146.7	1.7 – 3.2	100%	126.1 – 137.7	1.7 – 3.3	100%
177.6 – 195.1	0.5 – 2.6	100%	175.8 – 192.8	1.1 – 2.7	100%

TABLE II
EXPERIMENTAL RESULTS FOR CATCHING THE MOVING OBJECT

goal, we conducted experiments which let the robot catch balls moving “semi-predictably”. Figure 11 shows the “air conveyor” set up. The lightweight balls are blown anticlockwise by the four fans, one located in each corner. We mounted glass walls on the edge of the conveyor to prevent the balls from moving off the conveyor. The strategy of the experiment is as follows. The robot is initialized at a fixed position, $x = -700$ mm, $y = -309$ mm, $z = 300$ mm, waiting for a ball to pass. After the ball moves into the area -720 mm $< x < -680$ mm, which approaches the position of the end-effector in x coordinates, the robot starts to follow the ball, but only in the x direction. When the ball passes the line $y = -309$ mm, the robot starts to follow the robot in both x and y directions. The ball generally moves in the same direction until it hits the glass, when it bounces and generally reverses part of its velocity. After a couple of image processing iterations, we assume that the object is moving near a constant velocity and implement the lunge described in the above equations. A video clip showing some of these experiments is available at www.ces.clemson.edu/a Hoover/workcell/. The setup proved highly effective, and the robot was able to intercept the object consistently at speeds of up to 900 mm/s, with a standard deviation of 150 mm/s in the neighborhood of the impact position. Higher speed intercepts are the subject of our ongoing research.

IV. CONCLUSION

In this paper, we present a generic timing model for a robotic system using visual sensing, where the camera



Fig. 11. Experimental setup for air conveyor.

provides the *desired* position to the robot controller. We demonstrate how to obtain the values of the parameters in the model, using our dynamic workcell as an example. Finally, we show how this timing model can be used to solve problems, using as an example the problem of our industrial robot intercepting a moving object. Along with constant velocity motion experiments, we also describe experiments using an “air conveyor” experimental set up, in which lightweight balls move in semi-predictable motions. The results of the experiments show that our model is highly effective, and generalizable.

V. ACKNOWLEDGMENTS

The South Carolina Commission on Higher Education seed funded this project during 2000-2001. The Stäubli Corporation partially donated a state-of-the-art RX-130 industrial manipulator. The U.S. Office of Naval Research currently funds the exploration of this technology to naval warehousing through the Expeditionary Logistics program. We gratefully thank all these organizations.

REFERENCES

- [1] S. Hutchinson, D. Hager and P. Corke, "A Tutorial on Visual Servo Control," *IEEE Trans. Robotics Automat.*, vol. 12, no. 5, pp. 651-670, Oct. 1996
- [2] J. Gangloff and M. F. de Mathelin, "Visual Servoing of a 6-DOF Manipulator for Unknown 3-D Profile Following," *IEEE Trans. Robotics Automat.*, vol. 18, no. 4, pp. 511-520, August 2002
- [3] N. Papanikolopoulos, P. K. Khosla and T. Kanade, "Visual Tracking of a Moving Target by a Camera Mounted on a Robot: A Combination of Control and Vision," *IEEE Trans. Robotics Automat.*, vol. 9, no. 1, pp. 14-33, Feb. 1993
- [4] K. Hashimoto, T. Kimoto, T. Ebine and H. Kimura, "Manipulator Control with Image-Based Visual Servo," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 2267-2272, Sacramento, California, April 1991
- [5] P. Corke and M. Good, "Dynamic Effects in Visual Closed-Loop Systems," *IEEE Trans. Robotics Automat.*, vol. 12, no. 5, pp. 671-683, Oct. 1996
- [6] P. Corke and M. Good, "Dynamic Effects in High-Performance Visual Servoing," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1838-1843, Nice, France, May 1992
- [7] J. Stavnitzky and D. Capson, "Multiple Camera Model-Based 3-D Visual Servo," *IEEE Trans. Robotics Automat.*, vol. 16, no. 6, pp.732-739, Dec. 2000
- [8] S. H. Kim, J. S. Choi and B. K. Kim, "Visual Servo Control Algorithm for Soccer Robots Considering Time-delay," *Intelligent Automation and Soft Computing*, Vol. 6, no. 1, pp. 33-43, 2000
- [9] N. Houshangi, "Control of a robotic manipulator to grasp a moving target using vision," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 604-609, Cincinnati, Ohio, May 1990
- [10] P. Allen, A. Timcenko, B. Yoshimi and P. Michelman, "Automated Tracking and Grasping of a Moving Object with a Robotics Hand-Eye System," *IEEE Trans. Robotics Automat.*, vol. 9, no. 2, pp. 152-165, April 1993
- [11] F. Miyazaki, M. Takeuchi, M. Matsushima, T. Kusano and T. Hashimoto, "Realization of the Table Tennis task based on Virtual Targets," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 3844-3849, Washington, USA, May 2002
- [12] H. Nakai, Y. Taniguchi, M. Uenohara and T. Yoshimi, "A Volleyball Playing Robot," in *Proc. 1998 IEEE Int. Conf. Robotics and Automation*, Belgium, pp.1083-1089, May 1998
- [13] V+ User's Manual, Adept Corporation
- [14] Y. Liu, A. Hoover and I. Walker, "Sensor Network Based Workcell for Industrial Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1434-1439, Hawaii, Oct. 2001
- [15] Y. Liu, A. Hoover and I. Walker, "Experiments Using a Sensor Network Based Workcell for Industrial Robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 2988-2993, Washington, USA, May 2002
- [16] A. Hoover and B. Olsen, "A Real-Time Occupancy Map from Multiple Video Streams", in *Proc. 1999 IEEE Int. Conf. Robotics and Automation*, pp. 2261-2266, Detroit, May 1999
- [17] C. Hermanson, A. Hoover, M. Joseph, B. Judy, Y. Liu and I. Walker, "A Timing Model for a Dynamic Robotic Workcell," Technical Report, Department of Electrical and Computer Engineering, Clemson University, November 2002
- [18] Y. Liu, A. Hoover, I. Walker B. Judy, M. Joseph and C. Hermanson, "A New Generic Model for Vision Based Tracking in Robotics Systems," To appear in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, October 2003