DYNAMIC WORKCELL FOR INDUSTRIAL ROBOTS

A Dissertation

Presented to

the Graduate School of

Clemson University

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

Electrical Engineering

---

by

Yanfei Liu

August 2004

Advisor: Dr. Ian D. Walker

July 16, 2004

To the Graduate School:

This dissertation entitled "Dynamic Workcell for Industrial Robots" and written by Yanfei Liu is presented to the Graduate School of Clemson University. I recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy with a major in Electrical Engineering.

Dr. Ian D. Walker, Advisor

Dr. Adam W. Hoover, Co-Advisor

We have reviewed this dissertation
and recommend its acceptance:

Dr. Darren M. Dawson

Dr. John Wagner

Accepted for the Graduate School:

ABSTRACT

In this dissertation, we present a novel approach to workcell design for enhanced performance of industrial robot manipulators, in which the workcell features a specially integrated sensor network. In contrast to traditional robot workcells, in which the role of environmental sensors (if present at all) is relatively inflexible and tightly focused on a specific task, the sensor network performs real-time dynamic sensing of the entire workcell. The system is designed to make use of "off the shelf" components, and to integrate with the existing industrial robot controller, in order to allow the manipulator to perform more dynamic operations in a less structured workcell. Details of the novel sensor networked workcell and experiments with an industrial robot are presented in this work. Based on this experimental platform, we conducted research in the following three aspects:

1. 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 the commanded goal). 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 St*ä*ubli RX-130 industrial robot manipulator, a network of six cameras for sensing, and an off-the-shelf Adept MV-19 controller. We present experiments to demonstrate how the model can be applied.

2. Allowing dynamic motions for payloads within a workcell is a fundamentally novel idea for practical industrial robots. The ability to handle payloads moving in semi-structured ways would significantly increase the potential markets for industrial robotics. In this paper, we propose a generic theory to model the dynamic intercept and manip-

ulation capability of vision based industrial robot systems. In order to verify the theory, we present experiments using our industrial workcell prototype to dynamically intercept and manipulate semi-randomly moving objects. We conduct experiments over 1000 runs with two different kinds of dynamic tasks, “scoop” and “trap.” The experimental results validate our theory.

3. When we describe dynamic manipulation above, we focus on the dynamic properties of the objects. In the last part of this work, we add another variability to the dynamic manipulation by using a flexible end effector.

Key words: camera network, industrial robots, workcell, timing model, dynamic manipulation

Dedication

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

Introduction

Industrial manipulators are increasingly being applied across a wide range of applications. A typical approach, for a given application, is to design a workcell around the manipulator, in a way that matches the capabilities of the robot to the needs of the task. The workcell may feature specialized simple sensors (for example contact sensors to signal events requiring new actions of the manipulators and/or surrounding peripherals) [1]. In some cases, more ”general” sensors such as cameras may be used to recognize the location of type of items to be manipulated. This mode has proven effective in a number of applications including the electronics and automobile industries.

However, the above approach is in fact quite restrictive, severely limiting the number and type of applications accessible to robot manipulators today. In essence, today’s robot manipulators ”succeed” because their environments are carefully engineered to present them with carefully structured and regulated environments. Many applications which potential customers of the robotics industry would like automated remain inaccessible due to the inability to ”engineer out” uncertainties and imperfections in the process, such as wires and textiles. Robot industry professionals would like a more effective way to integrate sensors into robot workcells to allow the accomplishment of more unstructured tasks.

Recent advances in the merging of sensors with manipulator systems, such as those suggested for visual servoing [2], offer promise for improving the effectiveness of robot workcells. However, the issue of sensing, in real-time, unexpected or hard-to-model environmental effects (such as those found in manipulating flexible objects) remains a key problem. We are investigating a novel solution to this problem. The key component of the system is a network of cameras embedded in the workcell which can provide the continuous visual guidance to the robot. Using this camera network, the manipulator can react in real-time to dynamic effects in the workspace, potentially opening up a significant new range of applications for industrial robots.

In the mechatronics lab at Clemson University, we have constructed a prototype to research the viability of this "many-eyed" workcell. In chapter 2, we describe solutions to the problems of camera network placement, multi-video stream processing, calibration, and difference imaging. We also present how to adapt a conventional industrial robot controller to react to real-time external sensory data, in translation and orientation.

Visual sensing for robotics has been around for decades, but our understanding of a timing model remains crude. In general, visual sensing robot systems have several delays. Here 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 the commanded goal). Compensating for the delays has been a key problem that many researchers produced many different solutions [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. However, they all just provided a method to accomplish a specific task for their specific system. Basically, they are not generalizable. Therefore, in chapter 3 we introduce a timing model for the system where the camera is used to provide the *desired* or *reference* position to the internal robot controller. We also applied this timing model to our prototype and let the manipulator intercept constant velocity moving objects.

While we use interception of constant velocity moving objects to validate our generic timing model, the goal of our research is actually aiming towards dynamic manipulation of more generally moving objects. The idea of dynamic manipulation has been around for decades. There are quite a few researchers who have developed the basic models and conducted experiments to investigate the potential for automating tasks, such as throwing, hitting, tapping , toppling, etc [26, 27, 28, 29, 30, 31, 33, 20, 19, 34]. However, as far as we know, allowing dynamic motions for payloads within a workcell is a fundamentally novel idea for practical industrial robots. The ability to handle payloads moving in semi-structured ways would significantly increase the potential markets for industrial robotics. Therefore, in chapter 4, we propose a generic theory to model the dynamic intercept and manipulation capability of vision based industrial robot systems. In order to verify the theory, we present experiments using our industrial workcell prototype to dynamically intercept and manipulate semi-randomly moving objects. We conduct experiments over 1000 runs with two different kinds of dynamic tasks, "scoop" and "trap."

Finally we concluded this dissertation in chapter 6.

Chapter 2

The System Architecture of Workcell and Robot Motion Planning

2.1 Camera Networked Workcell

Conventional industrial manipulation makes limited use of machine vision. A common implementation places a camera on the chassis of a robot near its end-effector. This placement allows close-up sensing of objects under manipulation. Another common implementation places a camera overlooking a pick-and-place position. This placement allows sensing to help initiate or relinquish a grasp, with the assumption that the motion in-between does not need observation. Such is the case, for instance, if the grasp is firm, the object is rigid, and nothing else is in the path of the transport.

We are building an advanced machine vision system for a modern industrial robot. The key idea is to use a network of cameras to continuously observe the entire workcell. By observing the entire manipulation, the network will provide feedback to adjust grasps, paths, and goal points. For instance, parts delivered on a hanging chain might be grasped in the presence of unpredictable sway. If a person or unexpected object entered the workcell, the robot might be halted. If a part was to be affixed during a heat-change process, the

location might be adjusted to compensate for shrinkage. Most importantly, all of these scenarios might be enacted using the same hardware configuration.

2.1.1 Robot

For this work, we chose a six-axis Staubli RX-130 industrial robot manipulator. The Staubli is an ideal choice since it is a relatively high speed and high precision robot, featuring a conventional robot controller. The group at Clemson have established a close working relationship with Staubli (whose U.S. headquarters are located in nearby Duncan, S.C.) in the past few years. To allow the results of the research to be applicable to conventional industrial robots, we elected to utilize the standard controller for the robot and "build on top" of the existing platform. Details of the additional features added to allow the robot to respond to interrupts from the sensor network are detailed in section 3.

2.1.2 Camera placement

We desire a camera network that can observe the entire workcell simultaneously. The natural configuration to satisfy this goal places the cameras on the boundary of the workcell, looking towards the robot. The boundary of a typical workcell is the revolution of a paraboloid. As an approximation, we constructed a cube surrounding the workcell. The cube is 350×350 cm wide and 370 cm tall, extending 50 cm beyond the horizontal extent of the workcell and 100 cm beyond the vertical extent of the workcell. This configuration allows a deeper vertical perspective, while somewhat limiting the required floorspace.

Thicker industrial framing (5×5 cm) on the four vertical edges supports thinner industrial framing (5×2.5 cm) on the four top edges of the cube. Two additional struts of the same size are mounted on top of the cube and can be slid back and forth (see Figure 2.1). Two sides of the top are anchored to nearby walls. A single Hirose cable carries power, video, and a syncing signal to each camera, and is embedded in the framing. Each camera

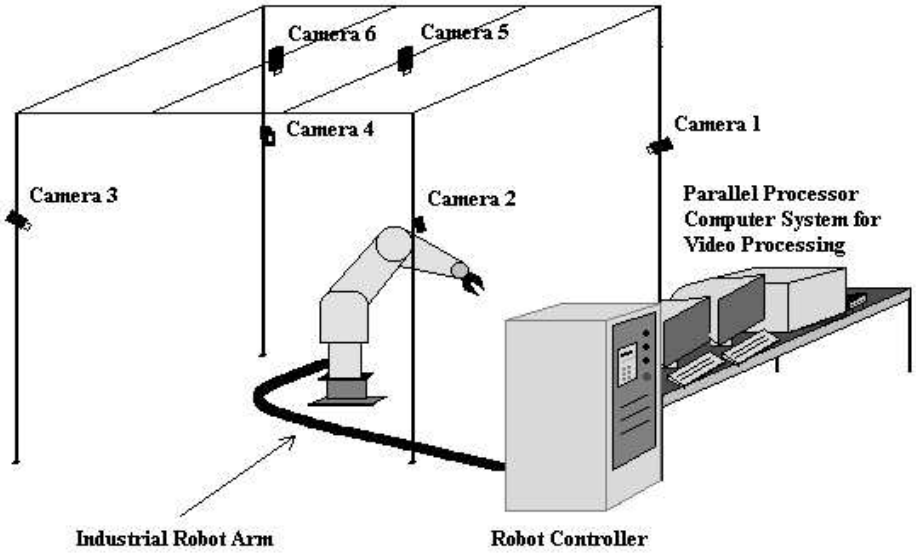


Figure 2.1: Workcell configuration.

weighs 140 grams, and does not significantly flex the structure. This configuration is rigid in the presence of moderate vibration such as could be expected on a factory floor.

The cameras can be placed on any part of the industrial framing. The mounts slide back and forth along each struct, so that a camera can be positioned anywhere on the top of the cube, and anywhere on the four vertical edges of the cube. This leaves the vertical planes of the workcell open for additional automation equipment, such as conveyors.

Sitting on one side of the workcell is the robot controller, power supplies, and computing hardware necessary to process the video feeds. Figure 2.2 shows a picture of our prototype.

2.1.3 Processing

Processing video from multiple cameras is inherently parallel. A shared multiprocessor (SMP) architecture provides an appealing platform for this problem. Preprocessing for the individual camera streams can be split among processors executing nearly identical code, while data fusion for a combined result accesses the same memory. At the time of this writing, 2-way SMPs are commonly manufactured for desktop use, and up to 8-way SMPs

Figure 2.2: Constructed prototype.

are commonly manufactured for server use. Our prototype uses an 8-way Compaq Proliant 8500R. We have observed near linear speedup on simple processing operations, such as image differencing (see Section 2.1.5).

Two Imaging Technology PC-RGB framegrabbers are used to capture images from six Sony XC-75 greyscale CCD cameras. A vertical syncing signal generated internally by one of the cameras is amplified and distributed to all the cameras. The six images are captured simultaneously using the trick of wiring each monochrome signal to one of the red, green, or blue inputs of each of the two framegrabbers.

Two limitations in our prototype are apparent. First, the chosen computer does not have an AGP bus, so that image display must use the PCI bus. Image display therefore competes with the throughput of framegrabbing and main memory access for processing. Second, the cameras and framegrabbers operate at the standard NTSC rate of 30 frames per second. This introduces a minimum delay of 33 ms between ”reality” and ”reaction”.

In [3], we described an algorithm for computing a real-time occupancy map from multiple video streams. The prototype demonstrated in that work acheived a throughput of 5

Hz, creating a 2D map from 4 cameras using a uniprocessor system. Our current work is focused on extending the algorithm to a 3D map, and expanding the number of cameras and processors. We are predicting that our prototype described herein will be able to create an $80 \times 80 \times 80$ map of space in the workcell at 30 Hz.

2.1.4 Calibration

Camera calibration is the process of determining a camera's internal parameters, such as the focal point and lens distortion, and the external parameters, such as position and orientation. These parameters model the camera in a reference system in the space being imaged, often called world coordinates. Once calibrated, a camera's 2D image coordinates map to 3D rays in world coordinates.

The standard camera calibration process has two steps. First, a list of 2D image coordinates and their corresponding 3D world coordinates is established. Second, a set of equations using these correspondences is solved to obtain a camera model.

A calibration target is commonly used to establish correspondences. The target is constructed so that its identification in imagery is simplified. The target is placed in the scene so that its world coordinates are easily measured. Often the calibration target itself defines the world coordinate system. A common calibration target is a grid. Multi-camera systems can be calibrated to a common coordinate system by viewing the same grid [6].

Calibrating cameras to observe an entire workcell poses a unique problem. The robot is permanently mounted in the middle of the workcell, which is ideally where a calibration target would be placed. There does not seem to be a simple way to acquire a large number of calibration points by capturing a single image per camera.

To overcome this problem we use the end effector of the robot as the calibration target. The robot is moved to a number of positions, 64 in our prototype. At each position, the image-location of the end effector is added to a list of points for each camera. If the end-

effector is not visible in a view, that point is ignored for that camera. The world coordinates of each point are taken from the robot controller.

Locating an arbitrary end-effector in an arbitrary image is not trivial. For instance, painting a target with a bright color is not sufficient. Figure 2.3 shows an example image from one of the cameras in our prototype. No provisions were taken to clear clutter from the background. Shrouding the workcell under dark cloth, and using controlled lighting, would simplify this problem. However, we envision that this configuration would be difficult to maintain on a common factory floor.

Figure 2.3: Background clutter expected.

To overcome this problem we constructed a blinking light for a calibration target. A spherical lightbulb was painted blue to reduce surface glare and mounted to the end effector. A simple circuit alternates power to the lightbulb, turning it on and off at a fixed rate. Cumulative differencing of the scene robustly locates the image position of the lightbulb. Figure 2.4 shows an example view of the lightbulb powered on, powered off, and the cumulative difference image.

Figure 2.4: Differencing to locate blinking lightbulb calibration target.

After using our methods to establish a set of correspondences, we use the camera model and non-coplanar solution method introduced by Tsai [4, 7]. We have found it to be robust for a variety of industrial and commercial cameras and lenses, and a dependable implementation is publically available [8].

The entire calibration process for our prototype requires only a single start command from the operator, and completes in approximately 2.5 minutes.

2.1.5 Difference Imaging

Difference imaging is a common image processing operation. It quickly reveals the locations of objects that have shifted position. Difference imaging in an entire workcell poses a unique problem. The robot is constantly moving throughout the workcell. Ideally, a background image for differencing would be acquired while the robot was absent. This is obviously impractical.

To overcome this problem we create a background image using a collage technique. The robot is moved to a series of positions which reveal all visible portions of the background for each camera. At each position, the joint angles of the robot are read from the controller. Using these angles a model of the robot is constructed, portraying each of the major links as a cylinder. This model is then used via ray intersection to determine which pixels are obstructed by the robot, and which pixels see the background. The sequence of partial images is then combined to produce a single collage background. Figure 2.5 shows an example for the view depicted in Figure 2.3, created by combining four images.

The process for creating the collage background images for our prototype requires only a single start command from the operator, and completes in approximately thirty seconds.

2.2 Robot Motion Planning

The Staubli robot can, in principle, move to any position inside its workspace, just the ”move A” command (where A is an arbitrary point) in the standard controller. The execution of this command is simple and smooth. However this command requires that the robot finish a fixed task [5]. The execution of this command cannot be interrupted in the motion and allow the robot to move to some new position. From point of view of the work discussed here, the robot will be required to change its trajectory in real time based on the vision feedback, so we need real time interrupt programming of the Staubli robot. We use the Alter command (an additional option available as an addition to the existing standard V+ language) to implement this. The use of the Alter command is summarized below.

Alter ( ) DX, DY, DZ, RX, RY, RZ

When the Alter command is enabled, this instruction will be executed once during each trajectory circle (every 16 milliseconds). This command makes the robot translate DX, DY, DZ along the X, Y, Z axes and rotate RX, RY, RZ about the X, Y, Z axes. The parameters can be set based on the world or tool coordinate system. We choose the world coordinate system because it is fixed and thus more convenient. The Alter command is called inside a loop, which runs through each trajectory cycle to let the robot reach the (modified) target position. In theory this is a simple (and very beneficial) extra feature of the robot controller, however in practise several modifications had to be made to interrupt the robot effectively in real time.

Figure 2.5: Background collage.

2.2.1 Altering Position

To change (end effector) position using the Alter command , we need to set the first three parameters: DX, DY, DZ. ¿From the above, we note that Alter command only works in Cartesian space. However generating trajectories in Cartesian space is not desirable because, for example, parts of the generated trajectory may not lie within the workspace. A better approach is to generate the trajectory in Joint space. However, as the Alter command works in Cartesian space, the trajectory generated in the Joint space needs to be converted to Cartesian space. Thus the following procedure is used:

1. Inverse kinematics is used to obtain the joint angles of the beginning and ending points.

2. The intermediate joint angles, between these two sets of joint angles, are generated using the method described in the ”Trajectory generation” section below.

3. The forward kinematics is used to compute the Cartesian coordinates of every point corresponding to the calculated joint angles.

4. The deviation in the Cartesian space is the difference between the calculated Cartesian coordinates in the present step and the Cartesian coordinates calculated in the previous step. This deviation is used as the DX, DY, DZ input to the Alter instruction.

Trajectory Generation

The key objective of the trajectory generation procedure is to make the robot start and stop smoothly. Also, when interrupted, the robot should change trajectory smoothly and move to the new goal point. The trajectory generation can be divided into three cases, 1) smooth start, 2) in case of interruption 3) smooth stop. If the robot is not interrupted, the standard trajectory generation routine operates in case 1 and case 3. When there is interruption from the sensor network, the trajectory generation switches to a modified version to handle case 2, which makes the robot move to the new goal position smoothly.

case 1: $A =$ initial joint angles. $B =$ initial joint angles. $C =$ goal joint angles.

case 2: $A =$ current joint angles (when interrupted). $B =$ the predicted joint angles $t = t_{acc} + t$ based on the old trajectory. $C =$ new goal joint angles.

case 3: $A =$ current joint angles. $B =$ goal joint angles. $C =$ goal joint angles.

Formulae for Trajectory Generation

$$\Delta C = C - B$$

$$\Delta B = A - B$$

when $-t_{acc} < t < t_{acc}$

$$h = \frac{t_{acc}+t}{2t_{acc}}$$

$$q = \left[(\Delta C \frac{t_{acc}}{T_1} + \Delta B)(2-h)h^2 - 2\Delta B\right] h + B + \Delta B$$

$$\dot{q} = \left[(\Delta C \frac{t_{acc}}{T_1} + \Delta B)(1.5-h)2h^2 - \Delta B\right] \frac{1}{t_{acc}}$$

$$\ddot{q} = \left(\Delta C \frac{t_{acc}}{T_1} + \Delta B\right)(1-h)\frac{3h}{t_{acc}^2}$$

when $t \geq t_{acc}$

$$h = \frac{t}{T_1}$$
$$q = \Delta C h + B$$
$$\dot{q} = \frac{\Delta C}{T_1}$$
$$\ddot{q} = 0$$

where t_{acc} is a constant and $2t_{acc}$ is the time for smooth start and smooth stop. The smaller t_{acc} is, the faster the robot starts and stops. T_1 is the time for the motion from the beginning to the ending point. T_1 is set based on the distance between these two points to render the overall velocity constant.

Example

We demonstrate the effectiveness of the above scheme using an example. The initial joint angles are $(10, 50, 80, 90, 140, 70)$. The goal joint angles are $(-10, 120, 140, 0, 40, 20)$. The interruption occurs at $t = 0.7$ seconds. The new goal joint angles are $(50, 20, 40, -20, 150, 80.)$ (All joint angles are expressed in degrees). The resulting joint trajectories are shown in Figure 2.6.

2.2.2 Altering Orientation

For real-time alterations of end effector orientation, extra effort is required when using the Alter command. Euler angles are used to represent orientation for the Staubli industrial robot. However, the yaw, pitch and roll convention was used in our motion planning, so we need to transform Euler angles to corresponding yaw, pitch and roll. Having obtained the initial and goal yaw, pitch and roll, the intermediate yaw, pitch and roll between them are generated using the method described in the ”Trajectory generation” section above. The difference between the calculated yaw, pitch and roll in the present step and the yaw, pitch

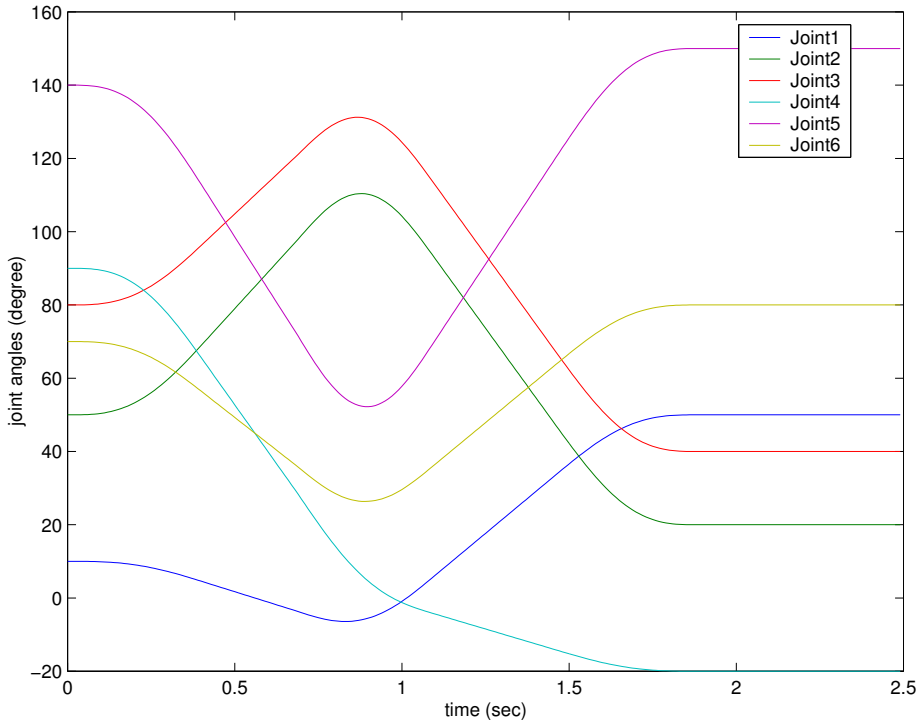


Figure 2.6: Interrupted Trajectory.

and roll calculated in the previous step, is used to compute the RX, RY, RZ input to the Alter command.

The usual definition for yaw, pitch and roll, implies the rotation matrix should be $R_{z\psi}R_{y\phi}R_{x\theta}$. However, the Alter command works in the reverse order $R_{x\theta}R_{y\phi}R_{z\psi}$. Here $R_{x\theta}, R_{y\phi}, R_{z\psi}$ are the rotation matrix for rotating θ around thte X axis, ϕ around the Y axis and ψ around the Z axis. Hence, we cannot directly use the deviation of the yaw, pitch and roll as RX, RY and RZ inputs to the Alter command. A transformation must be done first. In order to effect the transformation, two additional functions were implemented. Function #1 transforms $R_{z\psi}R_{y\phi}R_{x\theta}$ to yaw, pitch and roll and Function #2 calculates $R_{z\psi}R_{y\phi}R_{x\theta}$ from the given yaw, pitch and roll. Therefore we can obtain $[RM_i]$ (the desired rotation matrix) from the desired yaw, pitch and roll and $[RM_{i-1}]$ (the previous rotation matrix that the Alter command implemented) from the yaw, pitch and roll in the last loop. The following equation is then satisfied.

$$R_{x\Delta\theta}R_{y\Delta\phi}R_{z\Delta\psi}\left[RM_{i-1}\right]=\left[RM_i\right]$$

Now the problem is to compute $\Delta\theta, \Delta\phi, \Delta\psi$ given $[RM_{i-1}]$ and $[RM_i]$, where $\Delta\theta, \Delta\phi, \Delta\psi$ are the RX,RY,RZ input to the Alter command.

$$R_{x\Delta\theta\, y\Delta\phi}R_{z\Delta\psi}=\left[RM_i\right]\left[RM_{i-1}\right]^T$$

$$R^T_{z\Delta\psi}R^T_{y\Delta\phi}R^T_{x\Delta\theta}=\left[RM_{i-1}\right]\left[RM_i\right]^T$$

$$R_{z(-\Delta\psi)}R_{y(-\Delta\phi)}R_{x(-\Delta\theta)}=\left[RM_{i-1}\right]\left[RM_i\right]^T$$

Finally $-\Delta\theta, -\Delta\phi, -\Delta\psi$ are obtained using function #1.

Chapter 3

Generic Timing Model for Vision Based Robot Systems

3.1 Motivation

Figure 3.1 shows the classic structure for a visual servoing system [9]. 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 [10, 11, 12, 13, 14], 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 [10] developed a visual servoing system for a 6-DOF manipulator to follow a class of unknown but structured 3-D profiles. Papanikolopoulos [11] 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 [12] proposed a visual feedback control strategy for an eye-in-hand manipulator to follow a circular trajectory. Corke [13, 14] presented a visual feedforward controller for an

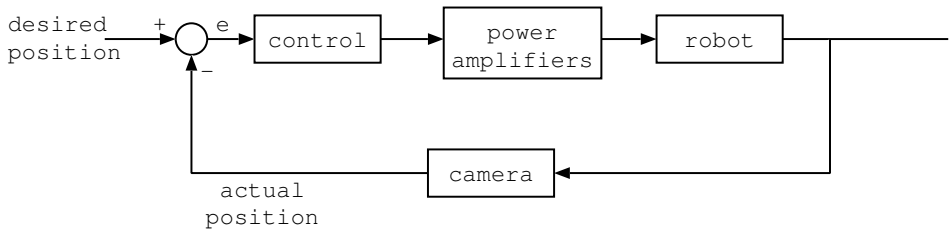


Figure 3.1: Classical visual servoing structure.

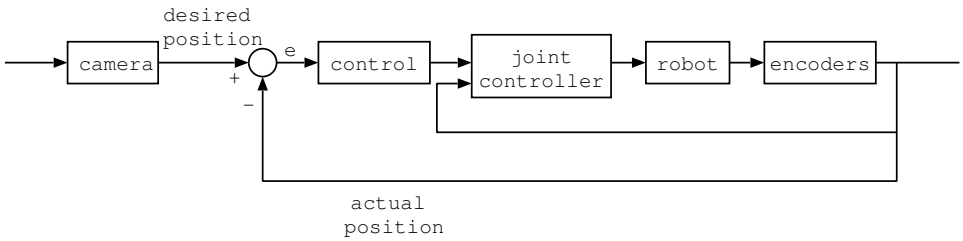


Figure 3.2: Vision guided control structure.

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 [15], 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, Stavnitzky [15] 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 [16], the camera is mounted over an environment to sense the actual position of the mobile robot as feedback to the controller. For example, Kim [16] 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.

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 work, we consider the problem where the camera is used to provide the *desired* or *reference* position to the internal robot controller. Figure 3.2 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 [17] 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 [18] demonstrated a PUMA-560 tracking and grasping a moving model train which moved around a circular railway. Miyazaki [19] built a ping pong robot to accomplish the ping pong task based on virtual targets. Nakai [20] 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 work, 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. The task of the robot is to intercept moving objects in real-time under the constraints of asynchronous vision and control. 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 the object state in reality is sensed and when the visual understanding of that object state (e.g. image tracking result) is available. We call this the *processing lag*.

work	image processing rate (HZ)	control rate (HZ)	processing lag (ms)	motion lag (ms)
Papanikolopoulos et. al. [11]	10	300	100	–
Hashimoto et. al. [12]	4	1000	–	–
Corke and Good [13, 14]	50	70	48	–
Stavnitzky and Capson [15]	30	1000	–	–
Kim et. al. [16]	30	30	90	–
Houshangi [17]	5	36	196	–
Allen et. al. [18]	10	50	100	–
Miyazaki et. al. [19]	60	–	–	–
Nakai et. al. [20]	60	500	–	–
this work	23	250	151	130

Table 3.1: Summary of related work

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 algorithm 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 work, 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 the object state 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 (the object state in 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 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 3.1 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 [18] 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 [10], 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 [10]. 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 [14] and Kim [16] 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 3.1, we notice that most real systems do not satisfy this condition.

Therefore, in this work we propose a continuous generic timing model to describe asynchronous vision-based control systems. This work is the first to explicitly model the motion lag of the robot and presents a general timing model for vision-based robotic systems. The approach is focused on improving real-time trajectory generation based on vision (Figure 3.2) and is independent of the control strategy applied.

The remainder of this chapter is organized as follows. In Section 3.2, 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 3.3, 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.

3.2 Methods

Figure 3.3 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 and when an estimate of the object state in reality is available. Similarly, the motion lag is the time between when the control command is issued and when the mechanical system finishes the motion.

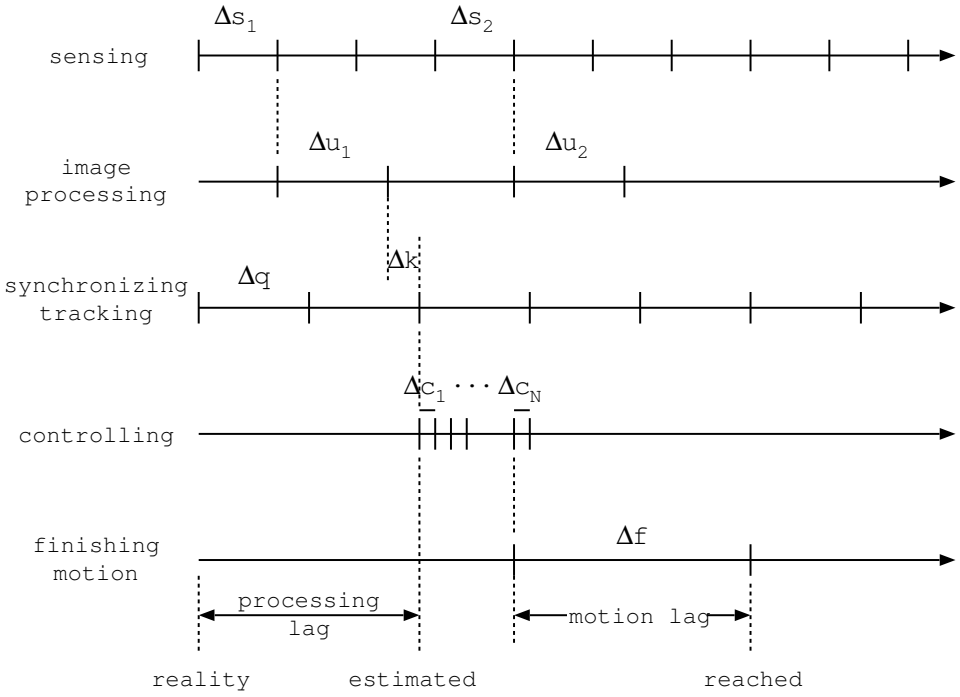


Figure 3.3: Timing model for estimating the lag and latency.

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 3.3, this is why Δs_2 starts on the next sync after the end of Δu_1.

Figure 3.3 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 3.4 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 3.4 as compared to Figure 3.3).

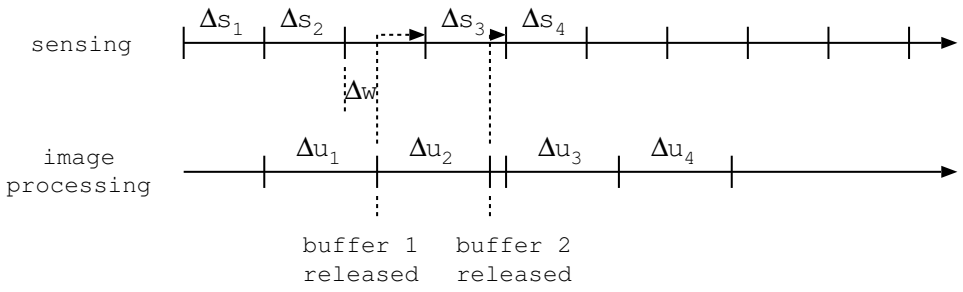


Figure 3.4: Timing model using double buffering for processing image data.

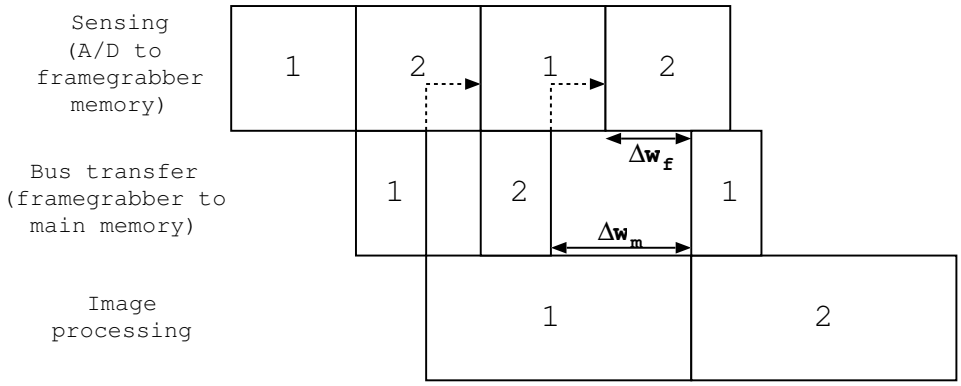


Figure 3.5: Timing model of using consecutive double buffering.

Figure 3.5 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 3.5 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 host memory for the completion of processing of the previous image. This is similar to the term Δw in Figure 3.4. 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 3.3) 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 3.4 - 3.5).

The fourth and fifth lines in Figure 3.3 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 3.3 depicts the case where control commands are cumulative (each new control command is relative to the last commanded goal). In Section 3.2.1 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. In Section 3.3, we derive an expression for implementing a "lunge" of the robot to intercept an object moving with an a priori unknown trajectory.

Figure 3.6: Our prototype dynamic workcell.

3.2.1 Prototype

Figure 3.6 shows a picture of our prototype workcell for this project. We use a St*ä*ubli RX130 manipulator with its conventional controller, the Adept Corporation model MV-19. 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 [22], we detailed the workcell configuration, calibration, image differencing and real-time robot motion planning. In [23], we presented some tracking experiments to show that our system can track different kinds of objects using continuous visual sensing.

Figure 3.7 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 measure-

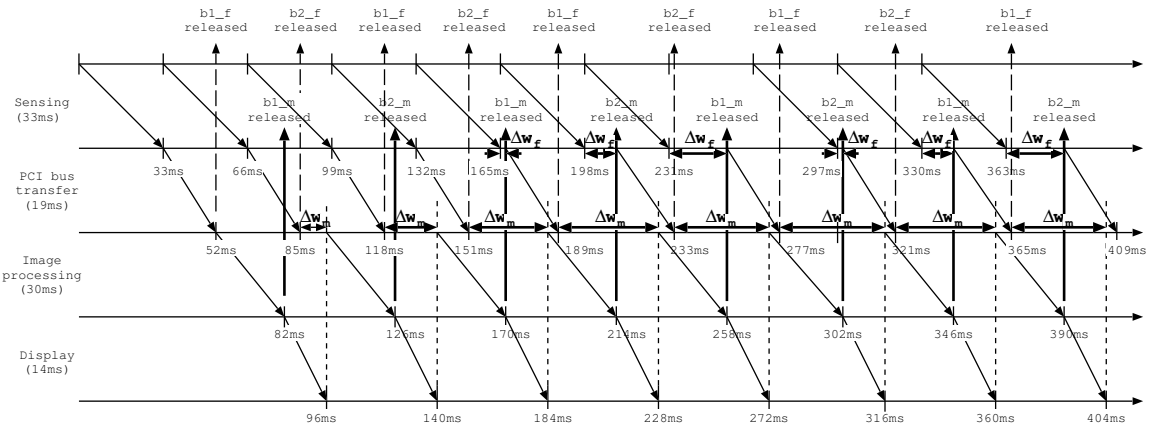


Figure 3.7: Information flow through system.

ments taken while the system was operating. We will discuss each part in detail in the following paragraphs.

Figure 3.7 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 format, so that for this system $\Delta s = 33$ ms. The object state in reality that is imaged takes 33 ms to transfer from the camera to the framegrabber. 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. 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 bytes) / (4 bytes $\times$ 33 MHz) = 19 ms, where the bandwidth is the theoretical maximum provided by the 32-bit 33 MHz PCI standard.

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 [3]. 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 3.5). The waiting time can vary depending on the phasing of Δs and Δu as shown in Figure 3.7. 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 3.7, 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 + Δk = (151 + Δk) ms. To unify the terms, we use Δl to express the computable partial lag time (151 ms).

After synchronization, the state of the object 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 Δq = 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 [21]. 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. 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 observed that the time 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. Our estimates for all parameters except for occupancy map computation time, display time, and robot motion lag, were generated via analysis of models based on known timing constraints, as discussed above. The remaining three parameters listed above required information not directly available to us, and were obtained from averages via empirical measurement. 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 3.8 shows a picture of the experimental setup. We set up another camera, completely external to our system, to observe its operation. 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

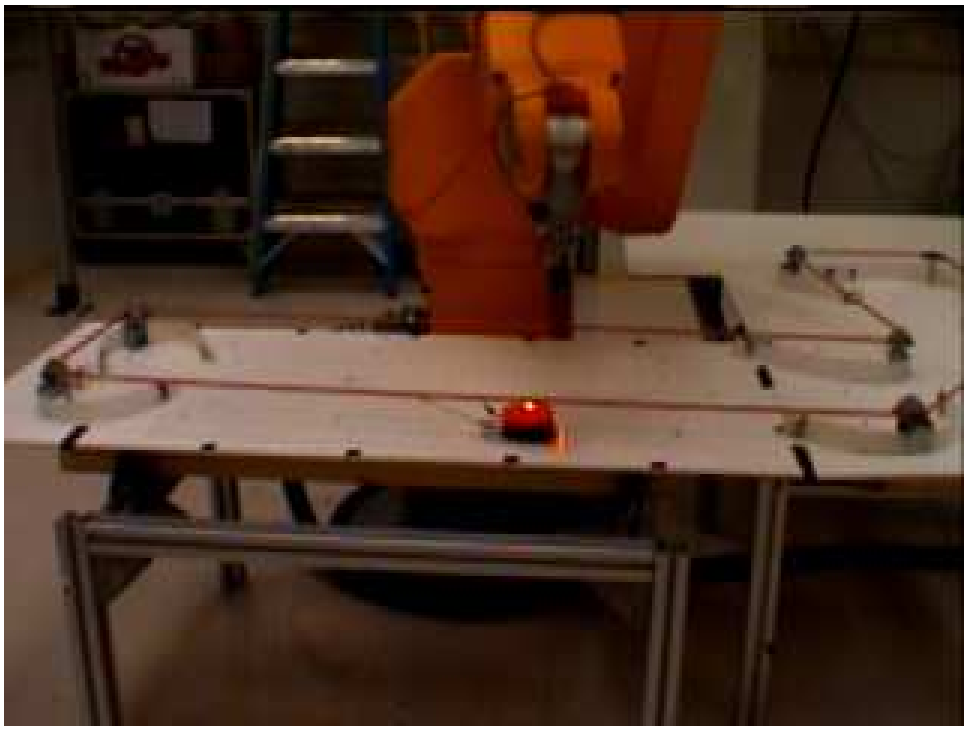


Figure 3.8: Experimental setup for lag measurement.

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 [24].

3.3 Experiments

In order to test our methods, we experiment with the problem of catching a moving object. Figure 3.9 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 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 $\vec{x}_{t-\Delta l-\Delta k}$.

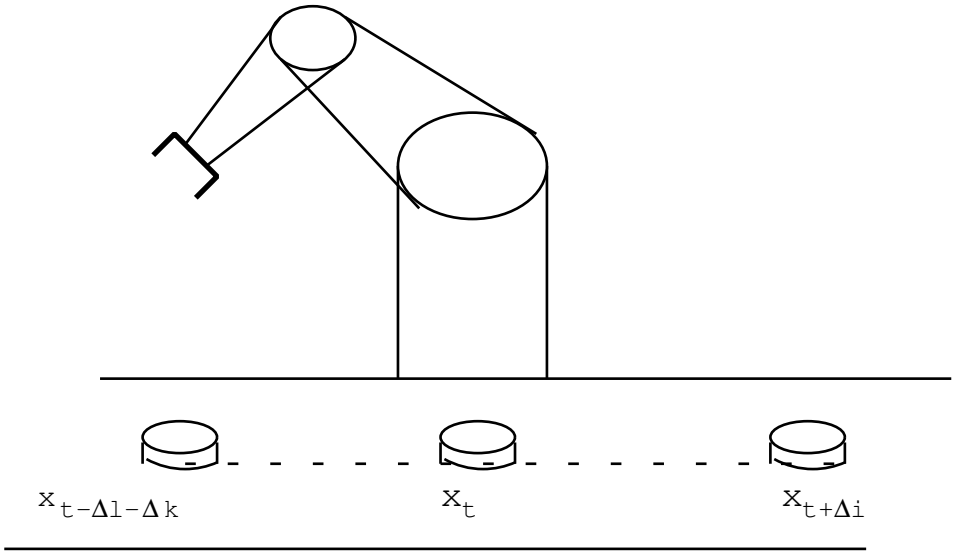


Figure 3.9: Scenario for intercepting objects.

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:

$$\vec{x}_t = \vec{x}_{t-\Delta l-\Delta k} + \vec{v}_{t-\Delta l-\Delta k}(\Delta l + \Delta k) \tag{3.1}$$

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:

$$\vec{x}_{t+\Delta i}[i] = \vec{x}_t[i] + \vec{v}_t[i](\Delta c \times (N-1) + \Delta f) \tag{3.2}$$

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:

$$\vec{x}_{t+\Delta i}[i] = \vec{x}_t[i] + \vec{v}_t[i](\Delta c \times (\vec{n}[i]-1) + \Delta f) \tag{3.3}$$

$$|\ \vec{x}_{t+\Delta i}[i] - \vec{x}_{t-\Delta q}[i]\ | = \vec{n}[i] \times d \tag{3.4}$$

Combining Equations 3.3 and 3.4, based on the assumption that the object does not change velocity direction between $t \ldots t - \Delta q$, we obtain:

$$\vec{n}[i] = \left\lceil \frac{(\Delta f - \Delta c)\ |\vec{v}_t[i]| + |\vec{x}_t[i] - \vec{x}_{t-\Delta q}[i]|}{d - \Delta c\ |\vec{v}_t[i]|} \right\rceil \tag{3.5}$$

$\Delta q = 40$			$\Delta q = 80$		
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 3.2: Experimental results for catching the moving object

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

$$|\vec{v}_t[i]| < \frac{d}{\Delta c} \tag{3.6}$$

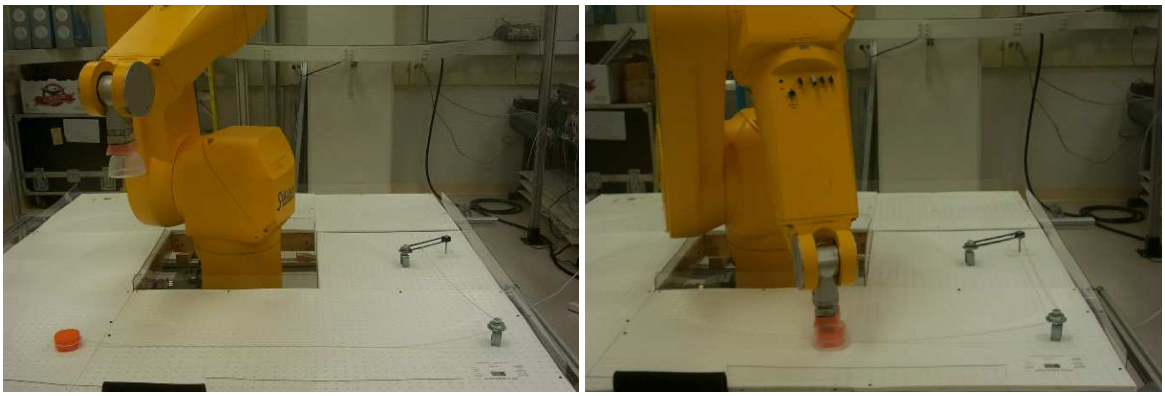
This equation represents the constraint for successful intercept between the velocity of the moving object and the key parameters in the timing model.

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

$$\Delta\vec{d}[i] = \frac{\vec{x}_{t+\Delta i}[i] - \vec{x}_{t-\Delta q}[i]}{N} \tag{3.7}$$

3.3.1 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 3.10 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.



(a) initial position (b) impact scene

Figure 3.10: Experimental setup for catching the moving 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 3.2 shows the results of the experiments. 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 demonstrate 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). We have additionally conducted numerous experiments with more unpredictable object trajectories (using fans to blow objects in the workspace in semi-predictable ways), and found the results to generalize well. These results will be reported in **??**.

Chapter 4

Dynamic Intercept and Manipulation of Objects Under Semi-structured Motion

Second journal paper

Chapter 5

Dynamic Manipulation Using a Novel Pneumatic Gripper

Grasp research using novel pneumatic gripper

Chapter 6

Conclusion

In this dissertation, we have discussed and detailed a novel approach to the design of a workcell for industrial manipulators. The system is based on an embedded network of cameras, which produce, in real-time, an occupancy map for the workspace. Using this map, the robot can react to manipulate flexible and dynamic objects in the workspace. In order to allow the robot to adapt to the sensor data without modifying the industrial controller, the trajectory generator was modified appropriately.

We also presented a generic timing model for a robotic system using visual sensing, where the camera provides the *desired* position to the robot controller. We demonstrated how to obtain the values of the parameters in the model, using our dynamic workcell as an example. Then we showed how this timing model can be used to solve problems, using as an example the problem of our industrial robot intercepting a moving object. The results of the experiments show that our model is highly effective, and generalizable.

Based on this timing model, we presented a novel concept, supported by generic and simple theory and extensive experiments to quantify the dynamic intercept ability of vision based robotic systems. We validated the theory by designing 15 sets of experiments using our industrial workcell. The results showed that the model was effective in predicting the capabilities of the workcell.

Bibliography

[1] S. Jorg, J. Langwald, J. Stelter, G. Hirzinger, C. Natale, "Flexible Robot-Assembly using a Multi-Sensory Approach", Proc. IEEE International Conference on Robotics and Automation, San Francisco, CA, 2000, pp. 3687-3694.

[2] G.D. Hager and S. Hutchinson, Eds., "Special Section on Vision-Based Control of Robot Manipulators", IEEE Transactions on Robotics and Automation, Vol 12, No. 5, October 1996.

[3] A. Hoover and B. Olsen, "A Real-Time Occupancy Map from Multiple Video Streams", in the proc. of *IEEE Int'l Conf. on Robotics and Automation*, 1999, pp. 2261-2266.

[4] R. K. Lenz and R. Y. Tsai, "Techniques for calibration of the scale factor and image center for high accuracy 3D machine vision metrology", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 10, no. 5, Sept. 1988, pp. 713-720.

[5] R.P. Paul, "Robot Manipulators: Mathematics, Programming, and Control", MIT Press, 1983.

[6] F. Pedersini, A. Sarti and S. Tubaro, "Multi-Camera Systems", in *IEEE Signal Processing*, vol. 16. no. 3, May 1999, pp. 55-65.

[7] R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3d vision metrology using off-the-shelf tv cameras and lenses", in *IEEE Trans. on Robotics & Automation*, vol. 3, no. 4, 1987, pp. 323-344.

[8] R. Willson, "Tsai Camera Calibration Software", http://www.ius.cs.cmu.edu/afs/cs.cmu.edu/user /rgw/www/TsaiCode.html.

[9] 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

[10] 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

[11] 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

[12] 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, Scramento, California, April 1991

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] V+ User's Manual, Adept Corporation

[22] 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

[23] 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

[24] 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

[25] Y. Liu, A. Hoover, I. Walker B. Judy, M. Joseph and C. Hermanson, "A New Generic Model for Vision Based Tracking in Robotics Systems," in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 248-253, Las Vegas, October 2003

[26] K. Lynch and M. Mason, "Dynamic Manipulation With a One Joint Robot," in Proc. IEEE Int. Conf. Robotics and Automation, pp. 359-366, Albuquerque, New Mexico, April 1997

[27] K. Lynch, N. Shiroma, H. Arai and K. Tanie, "The Roles of Shape and Motion in Dynamic Manipulation: The Butterfly Example," in Proc. IEEE Int. Conf. Robotics and Automation, pp. 1958-1963, Leuven, Belgium, May 1998

[28] T. Sakaguchi and F. Miyazaki, "Dynamic Manipulation of Ball-in-Cup Game," in Proc. IEEE Int. Conf. Robotics and Automation, pp. 2941-2948, San Diego, CA, USA, May 1994

[29] W. Huang amd M. Mason, "Experiments in Impulsive Manipulation," in Proc. IEEE Int. Conf. Robotics and Automation, pp. 1077-1082, Leuven, Belgium, May 1998

[30] C. Zhu, Y. Aiyama, T. Arai and A. Kawamura, "Motion Characteristics in Releasing Manipulation," in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1969-1974, Takamatsu, Japan, October 2000

[31] K. Lynch, "Toppling Manipulation," in Proc. IEEE Int. Conf. Robotics and Automation, pp. 2551-2557, Detroit, Michigan, May 1999

[32] M. Moll and M. Erdmann, “Uncertainty Reduction Using Dynamics,” in Proc. IEEE Int. Conf. Robotics and Automation, pp. 3673-3680, San Francisco, California, April 2000

[33] A. A. Rizzi and D. E. Koditscheck, “Progress in Spatial Robot Juggling,” in Proc. IEEE Int. Conf. Robotics and Automation, pp. 767-772, Nice, France, May 1992

[34] X. Zheng, W. Inamura, K. Shibata and K. Ito, “A Learning and Dynamic Pattern Generating Architecture for skillfull Robotic Baseball Batting System,” in Proc. IEEE Int. Conf. Robotics and Automation, pp. 3227-3232, San Francisco, CA, April 2000

[35] L. Žlajpah and B. Nemec, “Control Strategy for Robotic Yo-Yo,” in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 248-253, Las Vegas, October 2003

[36] Y. Liu, A. Hoover,I. Walker, “A Timing Model for Vision-Based Control of Industrial Robot Manipulators,” To appear in IEEE Trans. Robotics Automation

[37] Y. Liu, A. Hoover, I. Walker B. Judy, M. Joseph and C. Hermanson, “A New Generic Model for Vision Based Tracking in Robotics Systems,” in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 248-253, Las Vegas, October 2003