Visual Servoing for a Non-Reactive Industrial Manipulator

By

Zui Tao

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

August, 2015

Nashville, Tennessee

Approved:

Professor Richard Alan Peters II

Professor Mitchell Wilkes

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisor Professor Peters for his continuous support of my graduate study and instruct of my research. During the past two years, his guidance helped me and I enjoyed all the time with him both as his student and as his teaching assistant. I could not have imagined having a better advisor for my graduate study at Vanderbilt.

Besides my advisor, I would like to thank my thesis committee: Professor Wilkes, for his insightful comments.

I thank my lab fellows for their great collaborations, for the days and nights we were working together, and for the awesome friendships.

Last but not the least, I would like to thank my family: my parents and my girlfriend for supporting me financially and spiritually throughout my graduate study and my life in general.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Inspired by the nature of human, visual sensing has proven to be very useful in the field of robotics. One of its many uses is for visual servo control of robot manipulation [1]. Generally, there are two approaches to visual servo control, image-based visual servo (IBVS) and position-based visual servo (PBVS). The control scheme for IBVS moves the manipulator to minimize the distance in the image stream between the image of the end effector and the image of the object. PBVS uses 3D reconstruction of the scene to estimate the position of the object and moves the manipulator to those 3D coordinates. [2].

This work describes a systematic approach to PBVS using a standard 6-axis industrial manipulator with a standard, non-reactive controller. IBVS generally requires a reactive controller so that position of the end effector can be changed continually and instaneously in response to the position error measured by the vision system. But most off-the-self industrial robot controllers are designed to be preprogrammed for repetitive point-to-point motion, which is decidedly not reactive. Once the arm starts moving toward a programmmed position, its motion cannot be preempted. That makes IBVS difficult. Nevertheless, visual servoing is inherently reactive. The controller's limitations require a servoing algorithm to break up the manipulator's trajectory into small segments. The motion steps are determined by having the vision system continually compute the 6-axis pose (position and orientation coordinates) of the target object and send that pose to the manipulator at the beginning of each motion. This thesis demonstrates that PBVS can work well with a standard manipulator and controller. The approach lends itself to parallel implementation because the vision system and robot controller are computationally independent and connect periodically through the passing of coordinates from the vision system to the robot. [2, 3].

The thesis also demonstrates that visual servo control can be done inexpensively with a standard PC workstation and a pair of webcams; expensive industrial machine systems are unnecessary. Off-

the-shelf consumer-grade cameras suffice. This opens visual servoing to a much wider range of applicaitons. Two Logitech webcams were used in this work and met our accuracy requirements.

Our approach to visual servo control has the following characteristics:

- Economic vision system and computation with consumer equipment;
- Easy to implement with tolerable calibration errors;
- Can be combined with other object detection methods;
- Sufficiently generic to work on a wide variety of robotic manipulation tasks.

1.1 Related Work

Visual servo control has been studied for well over two decades both as a topic in signal processing and in robotics. In early works [4, 5], a vision system was integrated as a feedback control module with object modeling. Visual based control was shown to be effective in conjunction with object recognition. IBVS and PBVS were introduced as two different approaches [1] and demonstrated to be feasible [6]. During the same time, significant advances in 3D reconstruction from stereo vision were made by the use of projective geometry. Performance evaluation metrics were also devised [7, 8].

In early 2000s, different techniques were well established for both IBVS and PBVS [9, 10, 2]. Given reactive control at the robot motor servo level, the two approaches were proven to be comparable in performance [11] (although that performance is sensitive to the system setup). One tutorial on VS [3], combined IBVS and PBVS as an advanced approach.

At the same time, stereo vision systems apart from robotics were extensively researched [12, 13]. Recent advances in 3D sensing devices have enabled researchers to improve computational performance. Laser scanners have become less expensive and cheap consumer products such as the Kinect have made 3D sensing widely available. [14, 15, 16, 17, 18].

Although both stereo vision systems and visual servo control have been well developed, they have been applied to robots that have been designed with such control in mind. There has not

been much work to bring VS control to the currently installed base of industrial robots. Our work enables standard industrial manipulators to acquire reactive behaviors. This could enable industry to use existing infrastructure while expanding its capacity for flexible manufacturing. The economic benefits of advanced manipulation without having to replace existing robots could be significant. The principles guiding this research have been threefold: economic (use inexpensive components and existing robots), systematic (develop software systems that extend the capabilities of current robots), and applicable (to new problems).

1.2 Problem and Solution

Our systematic approach is straightforward and could be implemented as two stand-alone subsystems: the stereo vision system and the robotic system. The stereo vision system handles 3D spacial reconstruction and object detection while the robotic system handles coordinate frame transformation and dynamic control based on 3D position.

The development of a stereo vision system was performed as follows: The intrinsic parameters of stereo webcams were obtained from camera calibration using the Spatial Vision software from Universal Robotics. Then wrote software using OpenCV library [19] to reconstruct 3D space from stereo webcams with the intrinsic parameters applied. To achieve accurate pose oriented object detection, we combined detection based on both color features and depth information. A simple approach to 3D coordinate estimation of a targeted object in the camera frame was to compute the center of mass of the combined features.

The kinematics of the robot were determined empirically since that information is not supplied by the manufacturers of standard industrial manipulators. The robots are designed to be programmed manually by actually moving the robot to a sequence of positions using a programming pendant. Position-based visual servoing is essentially equivalent to programming the robot on-the-fly during its manipulation tasks. That requires the simultaneous calibration of the robot with the vision system. The transformation between camera frame and robot base frame must be determined. The robot control program was written using ROS [20]. In our real-time system, a tar-

get's 3D coordinates in robot frame is computed by transforming the object position estimated with respect stereo vision system's coordinate frame. The robot control program is given coordinates of the target pose of the end effector. A message-passing communications technique is employed to connect the visual program in C++ and the control program in Python to ensure simultaneous correspondence and dynamic tracking behaviors.

To illustrate the process straightforwardly, we include a system flowchart in the following Figure 1.1. The whole system comprises three parts. The separate, concurrently-operating vision system and robotic system are connected by message passing protocol that includes frame transformation.

The paper is organized as follows. Stereo calibration, 3D reconstruction, object detection and tracking are introduced in Chapter 2. Chapter 3 describes the robotic kinematics, frame transformation between the robotic and the vision system, and simultaneous tracking control. Chapter 4 demonstrates the system's dynamic behaviour and analyzes the sources of error. An appendix is included, where we explain explicitly how to run the system in our test bed.

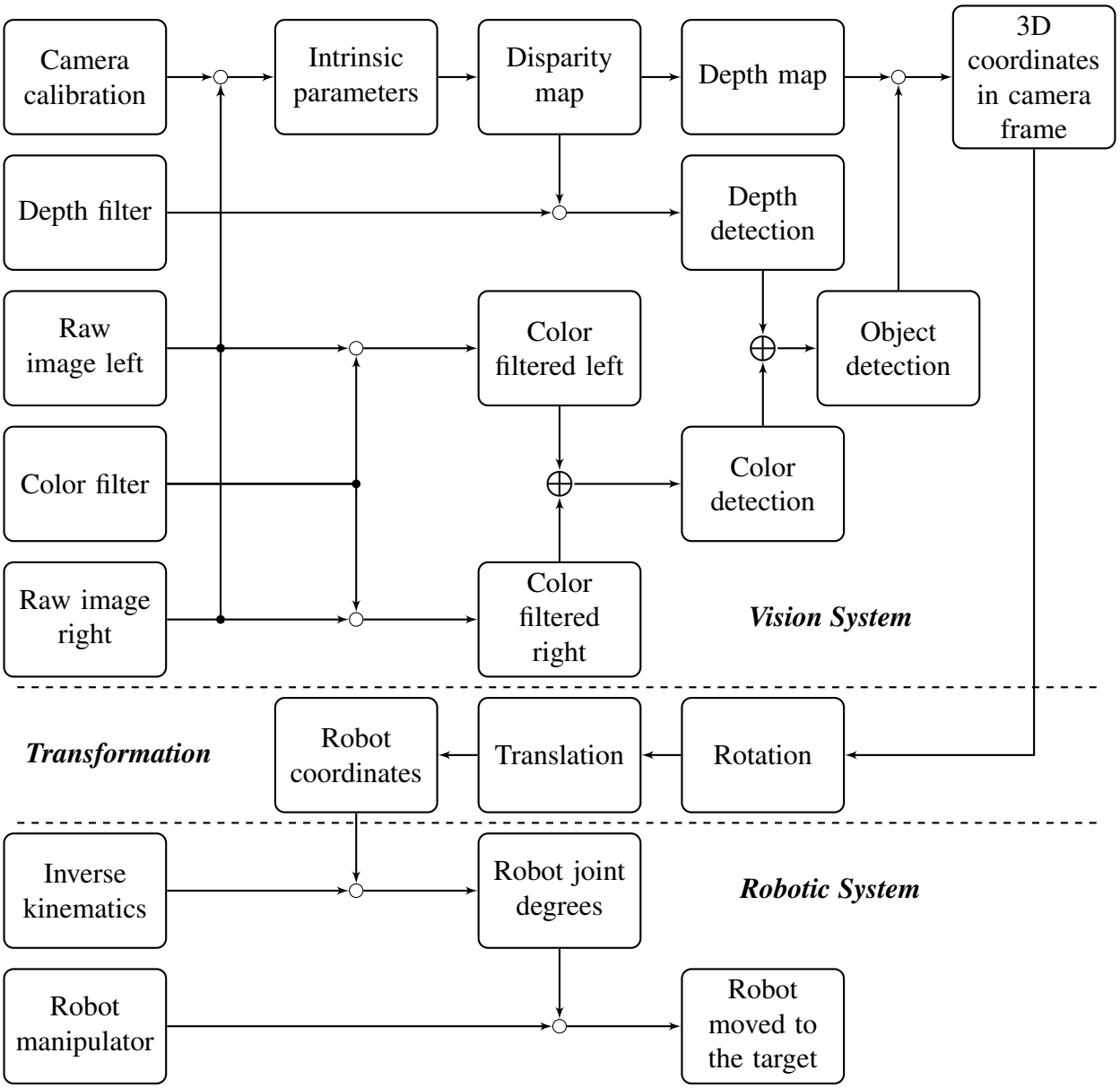


Figure 1.1: System flowchart

Chapter 2

STEREO VISION SYSTEM

We describe our system implementation as economic yet efficient, in part because we use off-the shelf webcams for the stereo vision system. Commercial 3D machine vision systems are very expensive; they can easily cost thousands of US dollars. We have found that Logitech webcams, although inexpensive are sufficiently accurate for our purposes. Its high-end product line features webcams the autofocus as well as the Carl Zeiss lenes which feeds sharp and fluid video with a high-definition resolution of 2-megapixels. Although these webcams are top-of-the-line consumer products, they are are sufficiently accurate to for visual servoing. In our system, we employ dual Logitech Pro 9000 that cost less than $100.00 USD, which significantly reduces the cost of our system in comparison.

These webcams do not, however have the built-in functions of commercial systems such as 3D construction, color detection, or object tracking. The work described in this chapter provides those capabilities.

We place our vision cameras above and facing the robot. The two webcams are placed at the same level. They are attached to a Kinect that is unused here but will be integrated in the near future. The lenes are adjusted manually roughly towards a point right behind the robot end effector with its up straight pose. This setting enables us to calculate correct depth information. It will be explained later in this chapter. The mean distance from the webcams to the robot is about 1.3 meters. That is the minimum distance for which all the robot motions are within the field of view. This also ensures that an object within the robot's reach is not too far for the vision system to detect and track. The completed system with dual webcams and the robot arm at the base is as shown in the following Figure 2.1.

OpenCV is an open-source library for real-time computer vision that has existed for more than a decade and is continually updated by users and volunteers. We used its optimized functions to

Figure 2.1: System setup

advance our code efficiency and visual servoing speed. PCL (Point Cloud Library) [21] is an open-source library for 3D geometric processing that has been under development since 2010. We use it for 3D reconstruction in this work. All the code for this work are written in C++ under Ubuntu 14.04 platform using Code Blocks as the IDE.

The rest of this chapter introduces the development of our vision system in three sections. In Section 2.1, we describe the intrinsic parameters of our webcams and estimate the transformation between the two cameras in the stereo pair. In Section 2.2, we calculate the image depth and reconstruct the 3D scenario based on calibration results using OpenCV and PCL. In the last Section 2.3, we describe our algorithm for combining depth detection and color detection to achieve accurate object tracking.

2.1 Calibration of Stereo Webcams

In this work, camera calibration can be separated into two parts. First, each of the two webcams is calibrated individually with camera matrix and distortion coefficients as its intrinsic parameters. Camera matrix consists of focal distances f_x, f_y and image center (c_x, c_y) as

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.1}$$

The distortion coefficient vector comprises radial distortion coefficients and tangential distortion coefficients.

Second, the stereo webcams are mutually calibrated. This step estimates the rotation matrix R and translation vector T between the coordinate systems of the cameras.

Universal Robotics, Inc. provided us with their commercial software Spatial Vision that combines the stero calibration process in a single program with a straightforward user interface. We set our webcam resolution to $640 * 480$ in the software and then conduct the calibration with 20 pairs of checkerboard pictures. Then we have the calibration results with the projection errors as shown in Figure 2.2.

Although we can tell the errors of our calibration results exceed the recommended values, with the median of 3D back-projection error still in millimeter scale, this set of results meets our requirement for our prototype system. We are also able to obtain the rectified image pair by the distortion coefficients as shown in Figure 2.3.

Example results of the rectification process are shown in Figure 2.3b. Rectification undistorts the images from stereo webcams and warps them to be parallel.

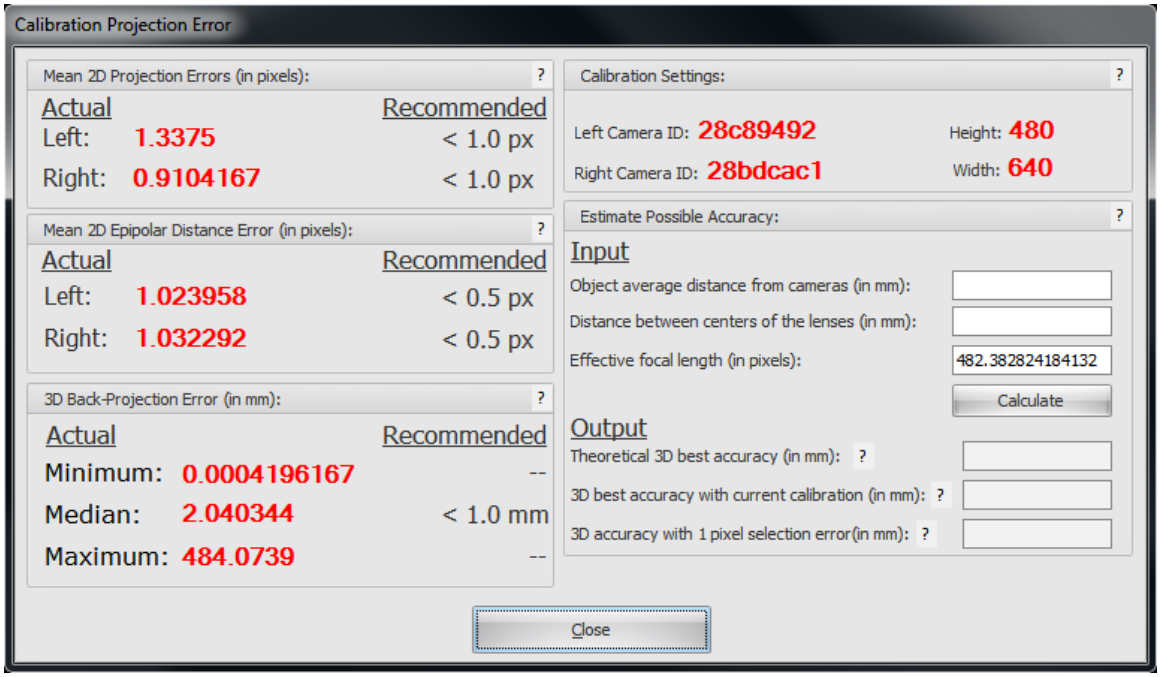


Figure 2.2: Calibration projection error

(a) Raw camera feed (b) Rectified image

Figure 2.3: Camera feed pairs

2.2 Stereopsis

After an sufficiently accurate camera calibration, we employ six result matrices in our stereopsis program. They include left camera matrix (M_0), right camera matrix (M_1), distortion of left camera (D_0), distortion of right camera (D_1), rotation (R) and transformation (T). These matrices

are taken as inputs to function stereoRectify(), which computes a rotation matrix for each camera to rotation camera images to the same plane. It produces five outputs which include

- Rectification rotation matrices (R_1, R_2) for left and right camera,
- Projection matrices (P_1, P_2) in the rectified coordinate system for left and right camera,
- Disparity-to-depth mapping matrix (Q) for later use when we reproject image to 3D space.

Then camera matrix, distortion, rotation and projection are inserted in function initUndistortRectifyMap() as inputs for each camera. This function computes the undistortion and rectification transformation. The outputs are represented in the form of maps for function remap(). The output map functions are then used in function remap() to undistort and rectify the grey image pair from our stereo cameras.

With the rectified stereo pair of grey images, we calculate the point-wise disparity which indicates stereo point correspondence and leads to depth information from the vision system. Since this work requires real-time correspondence, the calculation speed is important. We must choose the algorithm for the calculation of depth information to be sufficiently fast. We found that the block matching algorithm is a good choice because it produces about 7 frames of disparity map in a second with some accuracy traded. In the block matching algorithm, there are several parameters that had to be tuned manually. These play tremendously important roles in the accuracy of the disparity map. Our tuning process uses a program called Stereo BM Tuner and the following values in Table 2.1.

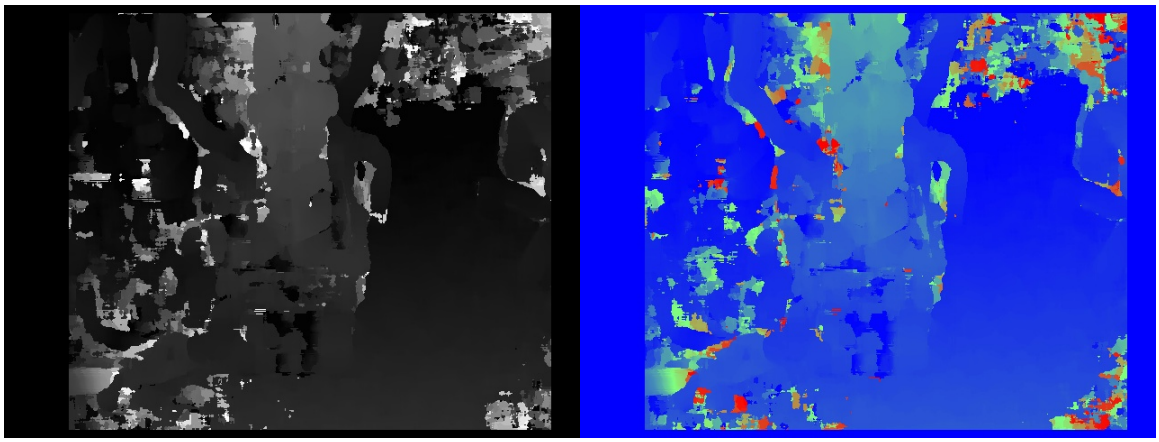
Notice that different systems require different sets of parameters. This set of parameters has been tuned for our system and might not work well in other stereo vision systems.

After the disparity is computed by using the BM algorithm with the BM parameters listed in Table 2.1, we normalize it for visualization. It contains the real disparity values reflecting the overlapping of the image pair. So we first normalize the real disparity values to the range from 0 to 255 which is a grey image showing the depth information. Then we convert the grey image

preFilterType	CV_STEREO_BM_NORMALIZED_RESPONSE
preFIlterSize	5
preFilterCap	63
SADWindowSize	19
minDisparity	-32
numberofDisparities	96
textureThreshold	0
uniquenessRatio	0
speckleWindowSize	0
spekleRange	0

Table 2.1: BM parameters

to RGB color disparity for better visualization. The grey and color disparity maps are shown as follows:



(a) Grey disparity (b) RGB disparity

Figure 2.4: Visualized disparity maps

Although the disparity map exhibits lots of noise, the depth information is well illustrated. In Figure 2.4a, the color changes from white to grey then to black with the distance increasing. In Figure 2.4b, the color changes from red to green then to blue with the distance increasing.

Recall that we have an disparity-to-depth mapping matrix Q as the output from stereoRectify(). We use that to reproject the disparity image back to 3D space. The function reprojectImageTo3D() takes real disparity and mapping matrix Q as inputs. It converts the single-channel disparity map to

a 3-channel image representing a 3D surface. For each pixel (x,y) and the corresponding disparity $d = disparity(x,y)$, it computes:

$$[X \quad Y \quad Z \quad W]^T = Q * [x \quad y \quad d \quad 1]^T \tag{2.2}$$

$$\_3\text{dImage}(x,y) = (X/W, Y/W, Z/W). \tag{2.3}$$

We found that each pixel in the 3-channel matrix _3dImage contains the real 3D coordinates in millimeters according to the camera frame. Although _3dImage would not be like a point cloud which is visualized as a 3D image, it is more usable for us to extract its 3D position information by each pixel location. This will be discussed more in detail in the next section.

2.3 Object Detection and Tracking

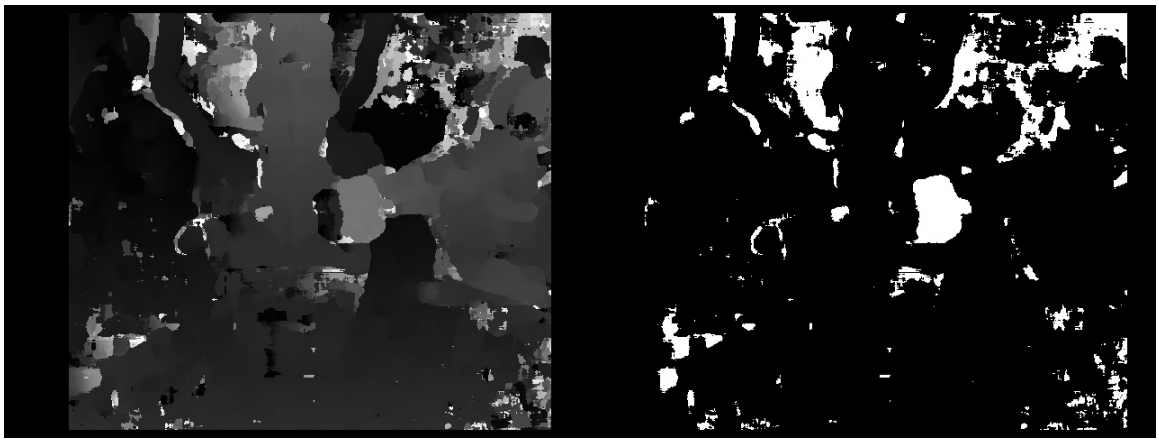
After disparity measurement, we obtain the stereo depth information in 3D space. In theory, we are able to extract the 3D position with respect to the camera frame of any point in the visualized disparity map. Under a real working conditions for a visual servo control system, we need our robotic arm to reach any visible object in its workspace. This was achieved by eliminating all the background and leaving a sufficient range of distances in the disparity map for the vision system to do the object identification.

Whereas the disparity map could only be used to determine the depth information, the color as well as the shape of an object are lost due to the overlapping of the stereo image pair. And if the robot arm reaches out randomly to anything within the range we set, it would not be a target oriented system. Worse, the robot arm could keep reaching toward itself once it moves into the recognition area. Therefore we devised a method, described in this section, that uses both color and depth information to detect and identify the object targeted by the stereo vision system. Shape information about the object will be left for our future work and will not be discussed here.

2.3.1 Depth detection

During depth detection, the vision system is not target oriented. It is a passive system that demarcates an area by distance around the stereo webcams and marks any object that within the area as a potential system target. The primary task here was to set up a region by distance for object detection.

Recall that we scaled the real-valued disparity map into the integer range from 0 to 255. We could simply filter out all the background depth information except for the area demarcated. So we set 100 as the threshold value to blackout the background. In return, we obtained the following threshold image with the original grey disparity map:



(a) Grey disparity (b) Threshold disparity

Figure 2.5: Threshold for disparity map

Unfortunately, this result was insufficient. One could see roughly from the original grey disparity map in Figure 2.5a that a person was sitting on the right side holding up a round object in the middle of the image. Figure 2.5b illustrates that the round object in the middle is well delineated after thresholding. But it is impossible for the vision system to tell which part is the servo target in the thresholded disparity map. Refer to the RGB disparity map in Figure 2.4b; it is apparent that the other stuff in the thresholded disparity map is noise coming from the original disparity map. For the vision system to eliminate efficiently as much noise as possible, a median filter was

applied to the disparity map. We found that a fairly large aperture linear size 105 for the median filter would remove the noise sufficiently. Then the results are as follows:

(a) Smooth grey disparity (b) Smooth threshold disparity

Figure 2.6: Threshold for disparity map with median filter

The result here shows that only the object is left in the image. We were able to estimate the object's depth information from this result.

After the successful object segmentation, we must estimate the 3D position of the target object. Each pixel in _3dImage contains its own 3D position. The detected object covers many pixels at the same time. We compute the mass center of the object from the smooth threshold disparity in Figure 2.6b and take the position of that pixel as the object's position for the robot arm to servo to. To compute the center of the object, we use spatial moments computed by:

$$m_{ji} = \sum_{x,y} (\text{array}(x,y) \cdot x^j \cdot y^i). \tag{2.4}$$

And the mass center $(\bar{x}, \bar{y})$ is

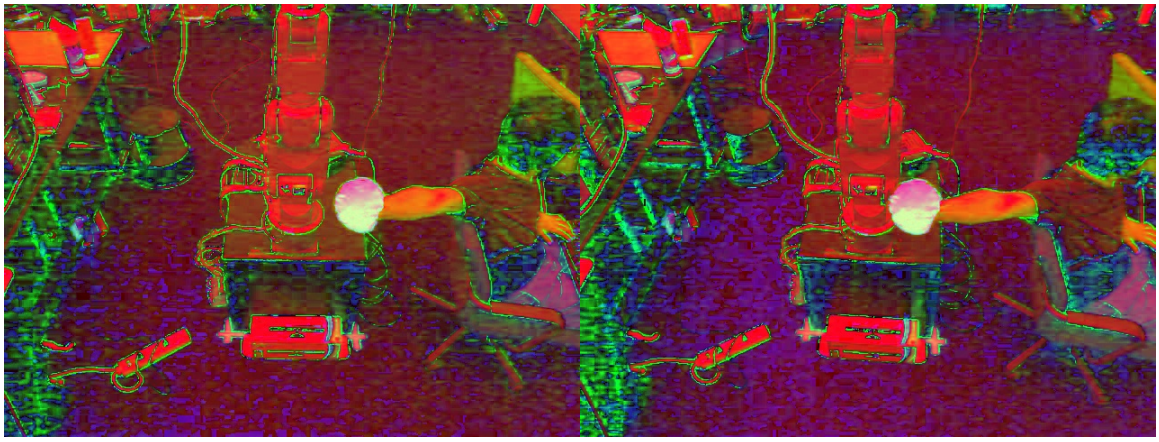
$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}. \tag{2.5}$$

Then the mass center $(\bar{x}, \bar{y})$ is the pixel location in _3dImage that is used to estimate the 3D position for the detected object.

2.3.2 Color detection

The vision system as described to this point cannot be described as a target oriented system because it does not have the ability to separate the target from other objects in the detection region. We use a color model of the object to disambiguate the objects found by depth detection alone. Color detection is straightforward; we convert the remapped RGB color image pair to HSV which is the color space most suitable for image segmentation. Most primary colors can be easily separated using only the value of Hue channel in HSV space.

Recall that we are not able to do color detection from the disparity map since it does not retain any color information from the raw feed stereo image pair. Therefore we must do the color detection directly from the stereo image pair. We convert the RGB image pair from our stereo webcams to HSV as follows:



(a) Left HSV image (b) Right HSV image

Figure 2.7: HSV image pair

In the HSV image, it is straightforward to extract a single color area using a suitable range of hue values. Here we used a red object and extracted the red area from the image pair. Notice that we converted the raw stereo image pair directly to HSV. Both the left and right images must be remapped for comparison after color filtering. We obtained the following image pair that indicated the red areas in Figure 2.8.

(a) Left colored area (b) Right colored area

Figure 2.8: Color filtered image pair

Here one can see that the results from HSV color filtering are good with the red object extracted in both webcams. There are some spurious areas in both images. They might be noise or other red stuff (real but not the target object) identified by webcams. We handle that via noise canceling and small area filtering since the spurious areas are small compared to the object.

To cancel the noise in the image pair in Figure 2.8, one can simply do an AND operand between the images in the pair since it is unlikely that the small noise regions will overlap significantly in both images. Moreover, overlap between the images will be likely to correspond to real objects that occupy the same position which could improve the upcoming centroid calculation. After the AND operation, the result of the color-filtered image pair is shown in Figure 2.9.

As shown in Figure 2.9, all the noise and other spurious areas are eliminated from the sum image. But we still find that the outlines of the object is not smooth. Therefore we apply a different median filter with radius of 21 to the images first. Notice that if we use only the color filter to detect the object, it will be detected if it was captured by stereo webcams. In most cases within the visible region, the robot arm would not be able to reach the object. Therefore, we combine depth detection with color filtering to improve the object detection. We applied a threshold of 50 to cover the entire robot move region within the grey disparity map (which was introduced in Section 2.3.1). Then we

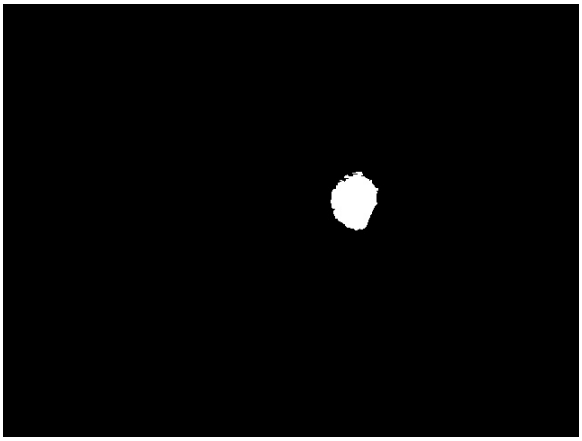


Figure 2.9: And operand of left and right images

do the AND operand between the thresholded grey disparity map and the smoothed color-filtered image sum. The resulting detection image is shown in Figure 2.10.

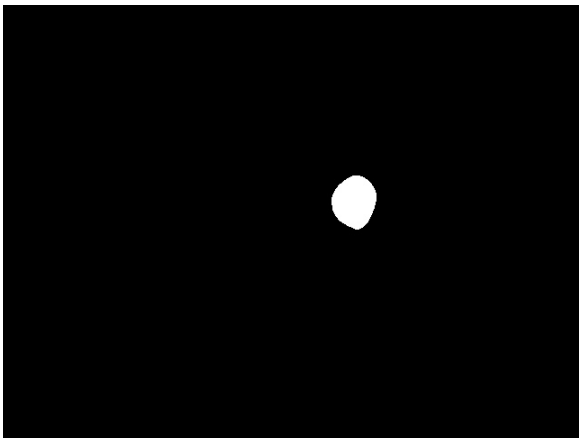


Figure 2.10: And operand of depth detected and color detected images

The tightness and appearance of the object detected result as shown in Figure 2.10 is satisfactory for servoing. We compute the center of mass in Equation 2.4 and 2.5 on that image. The centroid pixel $(\bar{x}, \bar{y})$ location in each image is taken as the object position. The procedure is sufficiently fast to track the position of a moving object (if it is not moving too fast). This will be further discussed in the next chapter.

Chapter 3

ROBOTIC SYSTEM

Our hardware testbed includes a Yaskawa-Motoman HP3JC, 6-axis industrial manipulator with an NXC-100 controller [22]. It is a small, lightweight robot[1] with a 3kg payload. It has a wide range of motor speeds from nearly undetectable motion to joint-dependent maxima in the range 150°/s - 450°/s. Each of its six motors is controllable with a positional accuracy of 1000 control pulses per degree. It is also highly repeatable; it will return to within ±0.03mm of the same location after hundreds of thousands of repetitions. In our experimental environment, the robot is placed 1.5 meters in front of the vision system in an upright pose so that it has as wide as possible range of motion within the stereo receptive field of the camera pair.

Visual servo control requires the mutual calibration of the robot and the vision system. The vision system estimates an object's position relative to the stereo camera frame. The robot moves its end-effector to positions relative to its base frame. Therefore visual servoing requires the transformation of object coordinates from the camera frame to the robot base frame.

Represent the transform from camera coordinates to robot base-frame coordinates by

$$\mathbf{x}^{\mathsf{r}} = B_{\mathsf{c}}^{\mathsf{r}} \mathbf{x}^{\mathsf{c}}, \tag{3.1}$$

where $\mathbf{x}^{\mathsf{c}}$ is the location of a point in space as measured in the stereo camera frame, and $\mathbf{x}^{\mathsf{r}}$ is the same point measured with respect to the robot's base frame. Transform $B_{\mathsf{c}}^{\mathsf{r}}$ is a rigid rotation plus translation, a member of the 3D special Euclidean transform group, SE(3). If $B_{\mathsf{c}}^{\mathsf{r}} \in$SE(3), then it has the form

$$B_{\mathsf{c}}^{\mathsf{r}} = \left[\begin{array}{c|c} R & \mathbf{t} \\ \hline \mathbf{0}^{\mathsf{T}} & 1 \end{array}\right], \tag{3.2}$$

where $R \in \mathbb{R}^{3\times 3}$ is such that $R^{\mathsf{T}}R = RR^{\mathsf{T}} = I$ and $\det\{R\} = 1$, (R is a rigid rotation), $\mathbf{t} \in \mathbb{R}^3$ is a

[1] The HP3JC is light for for an industrial manipulator – it weighs 27kg.

translation vector, and $\mathbf{0} \in \mathbb{R}^3$ is the zero vector. In terms of its parameters, $B_{\mathsf{c}}^{\mathsf{r}}$ is 12-dimensional. It has 9 entries in the rotation matrix and 3 in the translation vector.

Since the stereo camera pair and the robot are positioned manually, $B_{\mathsf{c}}^{\mathsf{r}}$ must be determined empirically. This is done by finding a single, specific point on the robot's end-effector in each of the two images of a stereo pair taken by the vision system. Since there are 12 parameters, the robot arm must be moved to at least 12 different positions within the vision system's stereo-optic field of view[2], and a stereo pair taken at each position. If the vision system has been calibrated properly, x^{c} can be computed from the disparity between the two images of the point on the end-effector. Point x^{r} is known since the robot was moved there by a control program. Given point pairs, $\{x_i^{\mathsf{c}}, x_i^{\mathsf{r}}\}_{i=1}^{N}$, Equation 3.1 can be solved for the transform parameters.

Thus the calibration procedure is to (1) move the robot to $N \geq 12$ positions, (2) take a stereo image pair of the robot at each of those N positions, and (3) estimate the transform parameters from the point pairs using an optimization method such as LMS estimation. (*cf.* Section 3.2.)

If the robot's position is to be controlled directly through is joint angles then in addition to $B_{\mathsf{c}}^{\mathsf{r}}$ the inverse kinematics of the robot must be known. Through inverse kinematics the joint angles can be derived from the Cartesian pose x^{r}.

In our control system, object detection and tracking are continual; the vision system outputs a time-series of measurements $\{x^{\mathsf{c}}(n)\}$. The time between measurement $x^{\mathsf{c}}(n)$ and $x^{\mathsf{c}}(n+1)$ will be on the order of milliseconds whereas motion from one pose, $x^{\mathsf{r}}(k)$, to the next, $x^{\mathsf{r}}(k+1)$, can be on the order of seconds. If all the points in the camera time-series were to be used, then a long queue of target points would accumulate while waiting for the arm to complete a motion. If the arm were to move to each successive point in the queue, visual servoing would bog down to the point of uselessness. Therefore we have included logic that prunes the visual point queue while preserving key points in the robot's trajectory. That is discussed in Section 3.3.

[2] 25 to 30 points are recommended to mitigate the effects of noise in the image measurements.

3.1 Forward Kinematics and Inverse Kinematics

A mechanical drawing of the Y-M HP3JC in its home position is given in Figure 3.1. The

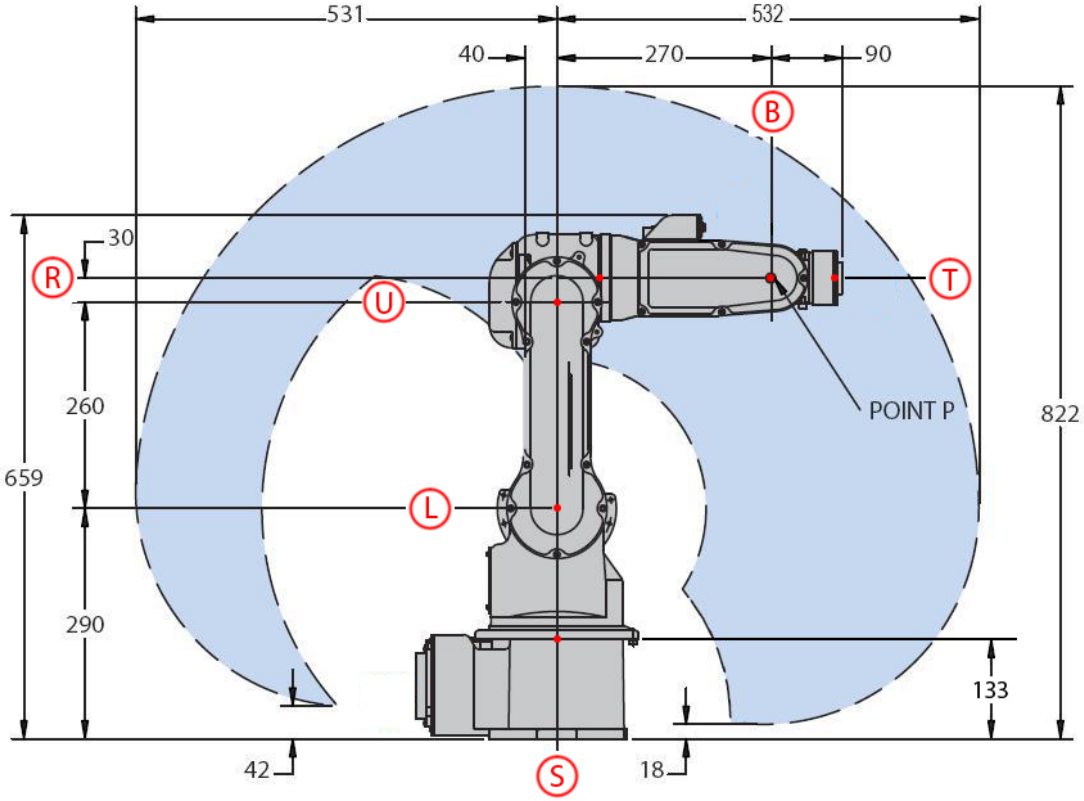


Figure 3.1: Dimensional specifications of the Yaskawa-Motoman HP3JC in side view

(Modified from http://www.used-robots.com/images/robots/original/hp3jc-side.jpg)

drawing has the locations of the six joints S, L, U, R, B, and T marked with red dots. The numbers indicate structural dimensions including the link lengths – the distances between adjacent joints. We assign the robot's base frame at the "S" joint (the base joint) which is 133 mm above the physical base of the robot. To place the robot's end effector at specific coordinates with respect to the base frame, we must determine the mathematical transform between the rotation angles of all the joints and the Cartesian coordinates of the robot's end effector. The mapping from joint angles to Cartesian pose is called "forward kinematics" and the retrograde mapping from Cartesian pose to joint angles is called "inverse kinematics" [23].

3.1.1 Forward kinematics

Forward kinematics give the Cartesian pose – position and orientation – of the end effector relative to the base frame as a function of the robot's joint angles. Forward kinematics can be defined in a number of ways. We use the Denavit-Hartenberg (DH) convention which represents the coordinate transform at each joint as the composition of four basic transformations [24],[25]. (*cf.* Equation (3.3).)

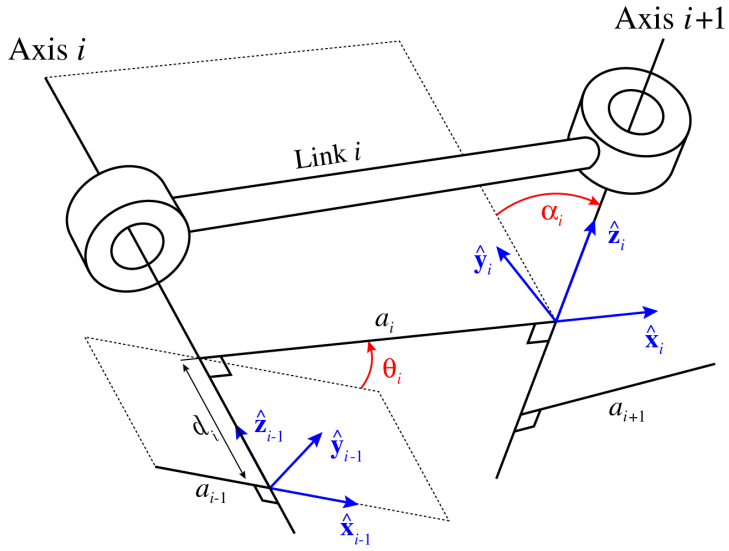


Figure 3.2: Link configuration in terms of Denavit-Hartenberg parameters

A *link* is the basic physical unit of an articulated manipulator (robot arm). It contains a motor that turns a single revolute joint and the hardware that connects its joint to the next one. An articulated manipulator is a connected chain of these links that extends from the physical base of the robot (link L_0, a special link that does not have a motor) through the first joint (link L_1) and successive joints to the end of the arm (link N) where a tool or end-effector is attached. Each of links $L_1, \ldots, L_N$ has an independently controlled joint so the entire chain (*i.e.* the manipulator itself) has N degrees of freedom (DoF).

In the D-H convention Link L_i has four parameters: a_i – the link length, α_i – the link twist, d_i – the link offset, and θ_i – the joint angle (*cf.* Figure 3.2). Link length a_i is the perpendicular distance between the axis of joint i and the axis of joint $i+1$. It can be envisioned as the length of the vector

between the axes that is perpendicular to both axes, *i.e.* the *mutual perpendicular* between link axes i and $i+1$. The link twist, α_i is measured in the plane defined by the mutual perpendicular. It is the angle in that plane between the projection of axis i and axis $i+1$, measured in a right hand sense around the perpendicular. Link offset, d_i, is the signed distance along joint i's axis from the point where a_{i-1} intersects the axis to the point where a_i intersects the axis. Joint angle θ_i is the angle between a_{i-1} and a_i measured around axis i.

At the end of each link, L_i, is a rectangular coordinate frame, $\mathscr{F}^i$. Its origin is at the intersection of the a_i perpendicular and the rotation axis of the *next* joint, $i+1$. Its z-axis, $\hat{\mathbf{z}}_{i+1}$, is coincident with the axis of rotation of joint $i+1$. In other words, coincident with the rotation axis of link L_i is the z-axis of link L_{i-1}. Coordinate vector $\hat{\mathbf{x}}_i$ points in the same direction as a_i and $\hat{\mathbf{y}}_i$ is constructed via right hand rule from $\hat{\mathbf{z}}_i$ to $\hat{\mathbf{x}}_i$. Frame, $\mathscr{F}^0$ is associated with the base of the robot and is called, therefore, the "base frame."

In summary the link parameters in terms of the link frames are:

a_i = the distance from $\hat{\mathbf{z}}_{i-1}$ to $\hat{\mathbf{z}}_i$ measured along $\hat{\mathbf{x}}_i$.

α_i = the angle between $\hat{\mathbf{z}}_{i-1}$ and $\hat{\mathbf{z}}_i$ measured around $\hat{\mathbf{x}}_i$.

d_i = the distance from $\hat{\mathbf{x}}_{i-1}$ to $\hat{\mathbf{x}}_i$ measured along $\hat{\mathbf{z}}_{i-1}$.

θ_i = the angle between $\hat{\mathbf{x}}_{i-1}$ and $\hat{\mathbf{x}}_i$ measured around $\hat{\mathbf{z}}_{i-1}$.

The first three parameters are constant for a revolute-jointed articulated manipulator and θ_i is the variable joint angle that sets the link's pose.

The coordinate transform, T_i^{i+1} from L_i to L_{i+1} is given by

$$
\begin{aligned}
T_i^{i+1} &= \mathbf{Trans}_{\hat{\mathbf{z}}_{i-1},d_i}\mathbf{Rot}_{\hat{\mathbf{z}}_{i-1},\theta_i}\mathbf{Trans}_{\hat{\mathbf{x}}_i,a_i}\mathbf{Rot}_{\hat{\mathbf{x}}_i,\alpha_i} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} R_i^{i+1} & \mathbf{x}_i^{i+1} \\ \hline \mathbf{0}^\mathsf{T} & 1 \end{array}\right],
\end{aligned} \tag{3.3}
$$

which is SE(3) like the camera to robot base coordinate transform. The first three rows of the fourth column of the matrix comprise $\mathbf{x}_i^{i+1}$, the 3-D position of of the next link in the chain, $i+1$, relative to the the position of the i^{th} link. The 3×3 matrix in the upper left of T_i^{i+1} is a rotation matrix, R_i^{i+1} that encodes the orientation of link $i+1$ with respect to link i's coordinate frame.

Link	a_i	α_i	d_i	θ_i
1 (S)	0	-90°	157	θ_1
2 (L)	260	180°	0	$\theta_2 - 90^\circ$
3 (U)	30	-90°	0	θ_3
4 (R)	0	90°	-270	θ_4
5 (B)	0	-90°	0	θ_5
6 (T)	0	0°	-90	θ_6

Table 3.1: DH table

The DH parameters for the Y-M HP3JC are given in Table 3.1. Links 1 through 6 include the base joint, S, the lower joint, L, the upper joint, U, and the wrist joints, R, B and T, "roll", "bend", and "tool". The DH parameters for each link specify a homogeneous transform[3] through Equation

[3]"Homogeneous" here refers to the 4-dimensional projective representation of points in $\mathbb{R}^3$. A vector $\mathbf{x}\in\mathbb{R}^3$ is represented as $\mathbf{x}_h = \begin{bmatrix} \mathbf{x}^\mathsf{T} & 1 \end{bmatrix}^\mathsf{T}$ in homogeneous coordinates. This enables rotation plus translation to be represented

3.3. The 6-axis robot, therefore, has 6 individual homogeneous transformation matrices ($T_i^{i+1}(\theta_i)$ for $i = 1,\ldots,6$). The elements of matrix T_i^{i+1} depend on angle θ_k. The transform matrix, H_b^e, that maps six given joint angles to the Cartesian pose of the end effector, is the composition of the homogeneous transformation matrices of each joint,

$$H_b^e(\theta_1,\ldots,\theta_6) = \begin{bmatrix} \text{Rotation} & \text{Translation} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R_b^e & \mathbf{t}_b^e \\ \mathbf{0} & 1 \end{bmatrix}$$
$$= T_6^e(\theta_6)T_5^6(\theta_5)T_4^5(\theta_4)T_3^4(\theta_3)T_2^3(\theta_2)T_1^2(\theta_1). \tag{3.4}$$

H_b^e is the forward kinematic transform matrix from base frame b (at the robot coordinate origin $\mathbf{0}$) to the end effector point e. Note that matrix multiplication proceeds from right to left. Vector $\mathbf{t}_b^e \in \mathbb{R}^3$ is x^r, the end effector position relative to the robot's base frame. The orientation of the end effector is encoded in the rotation matrix R_b^e.

3.1.2 Inverse kinematics

The mapping in the other direction – from Cartesian pose to joint angles – is the inverse kinematic (IK) transform. Whereas the derivation of the forward kinematics in the previous subsection was algebraic, the inverse kinematic transform is more easily derived from the robot's geometry. In this document, we have considered the inverse mapping from *position*, rather that pose, to joint angles θ_1, θ_2, and θ_3 – the base joint, S, and the lower and upper links, L and U, respectively. The S, L, and U alone are sufficient for the end effector to reach most of its workspace. (See Figure 3.1.) We have fixed joint angles θ_4, θ_5, and θ_6 to zero. This is sufficient for the work here since we are considering only the object's position and not its orientation in the servoing problem. The work will be extended at a future time to include object orientation. The base joint is easily solved for given the end effector coordinates $[x \quad y \quad z]^\mathsf{T}$ since base joint, S, is the only one that rotates

as a single matrix operation. Also, given a homogeneous transform Φ, if $\mathbf{y}' = \Phi(\mathbf{x}_h)$ is such that $\mathbf{y}' = [\ \mathbf{y}^\mathsf{T} \quad \gamma\ ]^\mathsf{T}$ where $\gamma \neq 1$ then $\mathbf{y}_h = \mathbf{y}'/\gamma$.

with respect to the vertical axis.

$$\theta_1 = \arctan\left(\frac{y}{x}\right). \tag{3.5}$$

Joints L and U are both rotating around horizontal axes so they can be solved together from the geometric configuration of ther robot as in Figure 3.3.

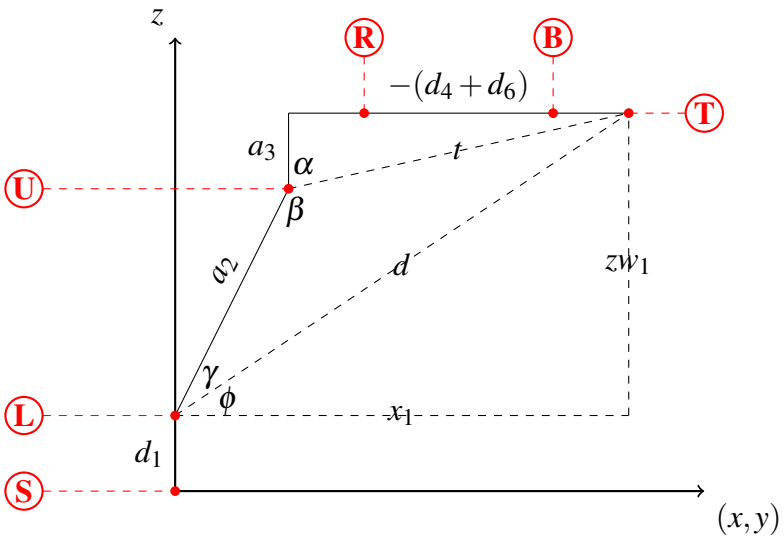


Figure 3.3: Geometric side view of the robot

Values d_1, a_2, a_3, d_4, d_6 are from the DH convention in Table 3.1. Vector $[x \quad y \quad z]^T$ contains the end effector coordinates. The rest of the labels in Figure 3.3 are described as follows: x_1 is the distance from the lower link L to the projection point of end effector on the same horizontal plane; zw_1 is the perpendicular distance from joint T to L; d is the distance from joint L to T; t is the distance from joint U to T; α is the angel between a_3 and t; β is the angle between a_2 and t; ϕ is the angle between d and x_1; γ is the angle between a_2 and d.

The following equations can be derived from the geometry in Figure 3.3.

$$x_1 = \sqrt{x^2 + y^2} \tag{3.6}$$

$$zw_1 = z - d_1 \tag{3.7}$$

$$d = \sqrt{x_1^2 + zw_1^2} \tag{3.8}$$

$$t = \sqrt{(d_4 + d_6)^2 + a_3^2} \tag{3.9}$$

$$\alpha = \arctan\left(\frac{-(d_4 + d_6)}{a_3}\right) \tag{3.10}$$

$$\beta = \arccos\left(\frac{a_2^2 + t^2 - d^2}{2a_2 t}\right) \tag{3.11}$$

$$\phi = \arctan\left(\frac{zw_1}{x_1}\right) \tag{3.12}$$

$$\gamma = \arccos\left(\frac{a_2^2 + d^2 - t^2}{2a_2 d}\right). \tag{3.13}$$

According to the joints configuration in Figure 3.3, the joint degrees for the lower and upper links, L and U, could also be derived so far.

$$\theta_2 = \frac{\pi}{2} - \gamma - \phi \tag{3.14}$$

$$\theta_3 = \alpha + \beta - \pi. \tag{3.15}$$

Substitute Equations 3.13 and 3.12 into 3.15 and substitute 3.10, 3.11 into 3.15, to obtain the inverse kinematics solution,

$$\begin{aligned}
\theta_1 &= \arctan\left(\frac{y}{x}\right) \\
\theta_2 &= \frac{\pi}{2} - \arccos\left(\frac{a_2^2 + [(x^2+y^2) + (z-d_1)^2] - [(d_4+d_6)^2 + a_3^2]}{2a_2\sqrt{(x^2+y^2) + (z-d_1)^2}}\right) - \arctan\left(\frac{z-d_1}{\sqrt{x^2+y^2}}\right) \\
\theta_3 &= \arctan\left(\frac{-(d_4+d_6)}{a_3}\right) + \arccos\left(\frac{a_2^2 + [(d_4+d_6)^2 + a_3^2] - [(x^2+y^2) + (z-d_1)^2]}{2a_2\sqrt{(d_4+d_6)^2 + a_3^2}}\right) - \pi \\
\theta_4 &= 0 \\
\theta_5 &= 0 \\
\theta_6 &= 0
\end{aligned} \tag{3.16}$$

3.2 Robot Calibration

As one might have noticed so far, in our inverse kinematics computation, $[x \quad y \quad z]^T$ are the coordinates in robot base frame which will not be fed directly from our vision system. Since object detection in the vision system only feeds back the desired coordinates of the end effector in camera frame, we have to figure out the coordinates of the same point in robot base frame. This process is called robot calibration in which we are going to find the mathematical transformation from camera frame to our robot base frame. Let us take our frame transformation as a rigid body rotation followed by a linear translation. According to Euler's Rotation Theorem [26]:

Theorem 1 *Any displacement of a rigid body that leaves one point fixed can be represented by a single rotation about an axis through the fixed point.*

Since any rotation can be defined by using three Euler angles $[\phi \quad \theta \quad \psi]$ as a result of a fix frame sequence of 3 basic rotations, the ZYX roll-pitch-yaw angles are obtained by $R = R_{z,\phi}R_{y,\theta}R_{x,\psi}$.

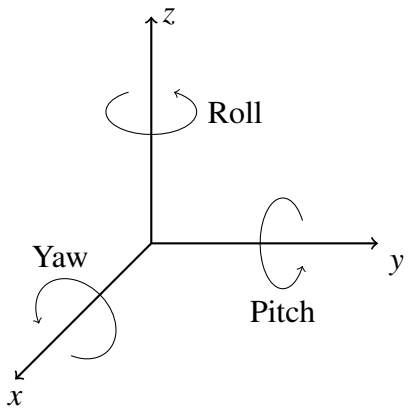


Figure 3.4: Roll-Pitch-Yaw (RPY) angles

Let us use $\mathbf{t} = [a \quad b \quad c]^T$ to represent zero point translation between camera frame and robot frame. We could have the following transformation equation:

$$\mathbf{x}^{\mathsf{r}} = R_{z,\phi}R_{y,\theta}R_{x,\psi} \cdot \mathbf{x}^{\mathsf{c}} + \mathbf{t}, \tag{3.17}$$

in which,

$$R_{z,\phi} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_{x,\psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{3.18}$$

And,

$$\mathbf{t} = \begin{bmatrix} a & a & & a \\ b & b & \cdots & b \\ c & c & & c \end{bmatrix} \tag{3.19}$$

Also, $\mathbf{x}^{\mathsf{c}}$, $\mathbf{x}^{\mathsf{r}}$ and T are 3×20 matrices. Let us refer back to Equation 3.1, then $B_{\mathsf{c}}^{\mathsf{r}}$ becomes

$$B_{\mathsf{c}}^{\mathsf{r}} = \left[\begin{array}{c|c} R_{z,\phi}R_{y,\theta}R_{x,\psi} & \mathbf{t} \\ \hline \mathbf{0}^{\mathsf{T}} & 1 \end{array}\right]. \tag{3.20}$$

At this point we are going to solve Equation 3.17 with 6 variables. So as many as 20 different sets of camera coordinates and robot end effector coordinates are needed to obtain satisfied transformation variables. Firstly we manually choose 20 different points based on the joint degrees and we have the following table of 20 different joint variables as shown in Table 3.2.

θ_i(radian)	01	02	03	04	05	06	07	08	09	10	11	12
θ_1	0	0	0	0	0	0	0	0	0.25	-0.25	-0.5	0.5
θ_2	0	-0.5	0.5	0.5	0.5	0.25	0.75	1	1	1	1	1
θ_3	0	0.5	0.5	1	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25
θ_4	0	0	0	0	0	0	0	0	0	0	0	0
θ_5	0	0	0	0	0	0	0	0	0	0	0	0
θ_6	0	0	0	0	0	0	0	0	0	0	0	0

13	14	15	16	17	18	19	20
0.25	0.25	0.25	0.25	-0.25	-0.25	-0.25	-0.25
0.5	0.5	0.25	0.75	0.5	0.5	0.25	0.75
0.5	1	1.25	1.25	0.5	1	1.25	1.25
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 3.2: Joint degrees

Then we move our robot according to these 20 different joint angles and save the corresponding color disparity maps from our vision system. The corresponding depth maps are in Figure 3.5 and coordinates of the end effector are in Table 3.3.

Their corresponding end effector coordinates are figured out by picking up the end effector points manually in Matlab using the funtion ginput(). Although the robot end effector seems to be obvious in the color disparity maps as shown in Figure 3.5, our way of finding those coordinates would probably introduce a certain amount of error to the final transformation matrices. In order to make our system simpler and easier to set up and calibrate, we are keeping this calibration method and its accuracy will be evaluated in Chapter 4.

CamCoords (mm)	01	02	03	04	05	06
x	57.3838	71.7439	47.3674	45.8052	53.5852	62.3344
y	-53.4536	-518.729	73.9183	-98.7286	-198.88	-373.178
z	1234.08	1276.42	1164.55	1074.84	1063.57	1109.01

07	08	09	10	11	12	13	14
57.8255	53.7214	195.823	-91.09	-249.277	368.398	201.809	168.796
-4.77649	167.29	164.784	140.133	100.319	91.4347	70.4248	-106.217
1065.6	1109.01	1121.27	1115.66	1156.11	1157.31	1181.81	1087.4

15	16	17	18	19	20
146.603	195.103	-70.6518	-69.7809	-6.58294	-105.468
-395.497	-25.8362	72.1086	-104.105	-371.764	-13.2935
1111.22	1081.08	1171.89	1087.4	1111.22	1085.29

Table 3.3: Camera coordinates

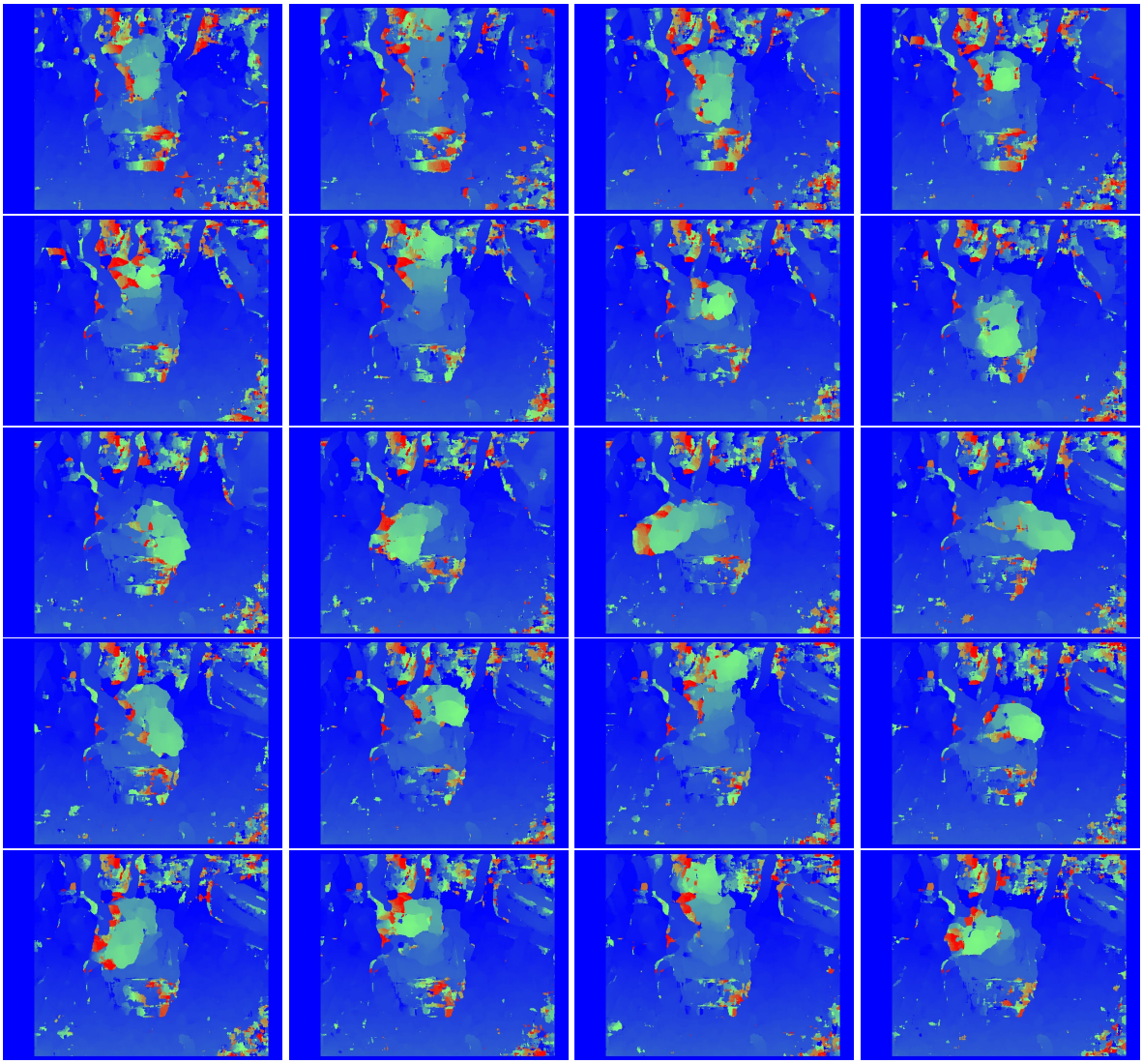


Figure 3.5: Depth maps

Finally we calculate the corresponding robot end effector coordinates using forward kinematics with the table of joint angles in Table 3.4.

RoboCoords (mm)	01	02	03	04	05	06
x	360	44.6141	484.6506	426.1976	367.6095	233.5897
y	0	0	0	0	0	0
z	447	704.3101	415.1715	584.0921	652.5121	728.0559

07	08	09	10	11	12	13	14
478.7730	560.1688	542.7545	542.7545	491.5944	491.5944	469.5840	412.9481
0	0	138.5880	-138.5880	-268.5592	268.5592	119.9045	105.4430
546.1598	415.6114	415.6114	415.6114	415.6114	415.6114	415.1715	584.0921

15	16	17	18	19	20
226.3280	463.8891	469.5840	412.9481	226.3280	463.8891
57.7910	118.4503	-119.9045	-105.4430	-57.7910	-118.4503
718.0559	546.1598	415.1715	584.0921	728.0559	546.1598

Table 3.4: Robot end effector coordinates

With these corresponding coordinates available in our case, Equation 3.17 becomes a redundant nonlinear equation group. Matlabs fsolve() method is used to solve non-linear equations to get the rotation matrix R and transformation matrix $\mathbf{t}$. The fsolve method requires the problem to be in the format of $F(x) = 0$, and thus we use the formula

$$R_{z,\phi}R_{y,\theta}R_{x,\psi} \cdot \mathbf{x}^{\mathsf{c}} + \mathbf{t} - \mathbf{x}^{\mathsf{r}} = 0 \tag{3.21}$$

in our calculations. The resulting equations for Matlab to solve is as follows: Where $[a \quad b \quad c]$ are the three variables in $\mathbf{t}$ and $[\phi \quad \theta \quad \psi]$ correspond to the three angular variables in R. Equations are written as a semicolon-separated list and assigned to the F variable. The right hand side of each equation (which is zero) is omitted.

With the solution as

$$[a \quad b \quad c \quad \phi \quad \theta \quad \psi] = [1274.7 \quad 40.7 \quad 1277.6 \quad 1.7 \quad 0.8 \quad 3.3], \tag{3.22}$$

in which $[a \quad b \quad c]$ are in millimeters and $[\phi \quad \theta \quad \psi]$ are in radius. Then we are able to use these values to command the robot to move to desired positions.

3.3 Simultaneous Correspondence

Since the image processing code is written in C++, and the code to move the robot is written in Python, we used an ROS node along with the publish-subscribe methods to let the two communicate with each other. ROS middleware handles all kinds of things for us, such as variable passing and reference stealing. First, we create an ROS node (with arbitrary name). When an image is processed successfully, the resulting camera coordinate is sent to that ROS node using the publish() method provided by ROS C++ API. For simplicity, the coordinates are sent as a semicolon separated string. Then, in the robot code (python), we use the rospy.wait_for_message([node_name]) to receive any camera coordinates published to the node we created. Then we use Python to process the message (camera coordinates) and hand the result to robot moving code. To convert the camera coordinates to robot coordinates, simply use the formula mentioned above in Equation 3.17.

So far we have calculated the R and T matrices. Thus we can compute the robot coordinates on the right hand side of the equation whenever we have received some camera coordinates. Then we use the inverse kinematics to compute the joint angles, which can be used by the robot moving code. In order to protect the robot, we have set some limits to avoid any collision.

Chapter 4

EXPERIMENT AND ANALYSIS

With the visual servo program and robot control program running at the same time, our system is tested and analyzed in this chapter. Due to the fact that this system is a prototype, our primary goal in this chapter is to demonstrate that our idea works and the system behaves exactly as we designed. We evaluate the effectiveness of our systematic approach from different aspects. First of all, system integrity and dynamic behaviour are demonstrated under a live environment with target object consistently changing its position. Error analysis will then be carried out for random unexpected movements of the Y-M HP3JC.

4.1 Dynamic Behaviour

In order to demonstrate the system dynamic behaviour, we chose a red object together with a color range filter for HSV values from $(150, 84, 130)$ to $(255, 255, 255)$. Using a red object here would suffer smaller color interference according to our laboratory environment and a better detection was obtained by a color filter with a wider HSV value range. Then roughly 10 minutes of continuous detection and tracking was done. We have 9 representative frames from the recorded video showing accurate detection and tracking performance as follows in Figure 4.1.

The red object we used in this experiment is about 3 inches in diameter and for most of the time the robot end effector succeeded to reach it on target when the object is moved to different places constantly. From the frames shown in Figure 4.1, it is obvious that our prototype system is working as we expected. Both object detection and dynamic tracking were achieved back and forth. Nevertheless, in order to further refine and improve our system to meet the needs for different kinds of tasks, we still need to find out the way to evaluate the system accuracy. Since we were only dealing with a big object with the diameter of 3 inches and the robot end effector is not in a small size, our system error would not be seen easily from the dynamic behavior. As a result, we

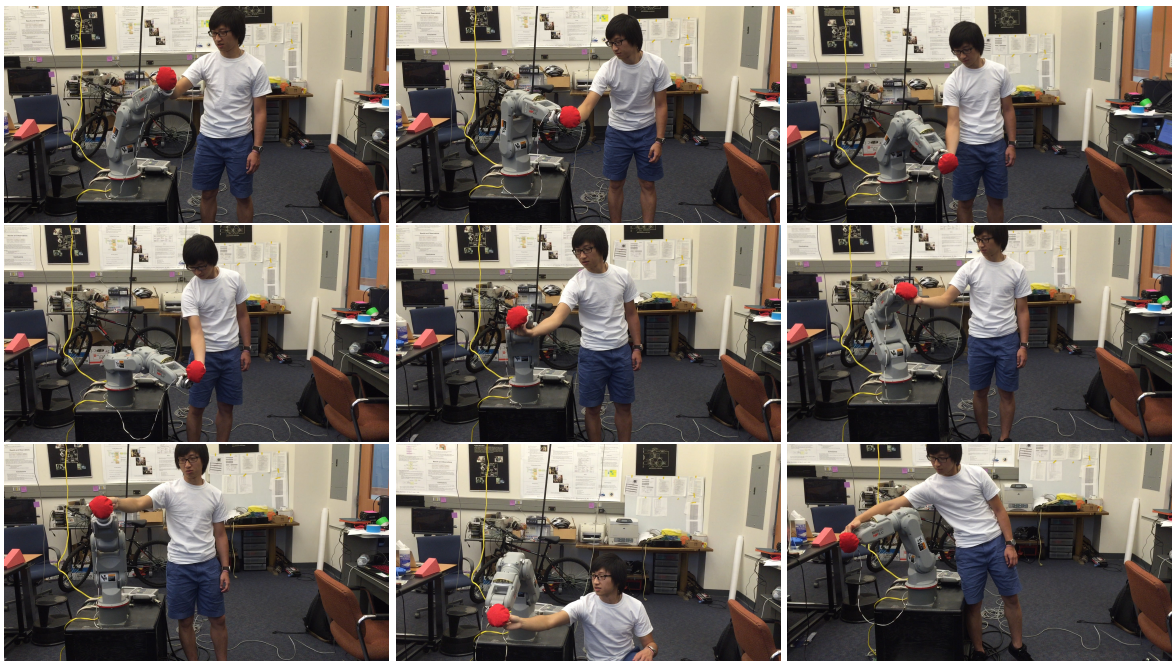


Figure 4.1: Capture and tracking

have to analyze our system error from the aspect of system components.

4.2 Error Analysis

When we look back to review our system, the developing process suffers from errors while keeping system simplicity. In this section we discuss three main error sources including camera calibration, depth map and robot calibration.

4.2.1 Error from vision system

Recall the camera calibration projection error in Figure 2.2, the 2D projection errors for both webcams are at least 1 pixel which leads to the median 3D back-projection error of 2 mm. Most of these calculated errors come from rotation between stereo webcams and len distortion [7]. Moreover, there are even greater undetermined intrinsic errors which are caused by sensor noise [8]. Nevertheless, block matching algorithm was used to ensure the calculation speed as compensation for matching errors of correspondent points [27].

When it comes to center point calculation for the detected object, there is a chance that the calculated center would be off the object. Recall the equations we used to calculate the mass center in Equation 2.4 and 2.5: the mass center would mostly lie in the area of the detected object with a regular shape as shown in Figure 2.10. If the detect object is not in a regular shape and the mass center might lie somewhere off the detected area. As a result, the depth information extracted from the depth map indicates a point in the background. In the worst scenario, if our calculated mass center lies in a noise area that we could find a lot in Figure 2.4, then there will be no solution for inverse kinematics or the robot arm will move to somewhere unexpected than our target position.

4.2.2 Error from robot calibration

Although there are lots of error sources from the vision system, they produce a small amount of errors in several millimeters which are pretty acceptable and it is very unlikely to happen with a fetal error. Most of our system errors exist and are caused by frame transformation between webcams and robot.

While we recorded 20 sets of correspondent points both in camera frame and robot frame to calculate the transformation, since we manually picked those end effector points in the color disparity map, each set of points is not perfectly accurate. Also, with 20 sets of imprecise points, in order to solve the redundant nonlinear equation group, nonlinear estimation was used to obtain approximate rotation and transformation matrices. As a result, we have to test this approximate solution with each point set to evaluate its accuracy. We substitute all the 20 point sets together with our approximate rotation matrix R and transformation matrix $\mathbf{t}$ into Equation 3.21. Then we have the following error table for all point sets in x, y and z directions of robot frame in Table 4.1.

Error (mm)	1	2	3	4	5	6	7
x	-12.6570	-50.3916	1.4705	5.1360	1.9158	-17.8688	23.0215
y	-38.7014	-36.0691	-39.1020	-33.3874	-25.9206	-24.8364	-19.0151
z	16.4698	65.3934	4.7227	22.4112	33.6268	52.2326	-1.9908
Mismatch	43.9233	90.0921	39.4136	40.5384	42.5006	60.5342	29.9254

8	9	10	11	12	13	14	15
29.7343	23.8664	36.6147	45.4502	-16.9955	-12.1219	-6.8387	-35.1772
-25.2813	-24.0296	-31.7496	-64.0326	12.4470	-7.4673	-18.0914	0.5432
-25.4092	-41.2920	-0.9193	10.3367	-24.3155	-14.6903	11.1686	71.3532
46.5715	53.4046	48.4718	79.2006	32.1717	20.4574	22.3339	79.5551

16	17	18	19	20
-0.0724	20.6156	16.0368	-5.0291	32.5546
-3.0526	-37.3222	-44.2783	-35.7591	-65.1425
-6.3641	8.6712	25.1879	54.1808	1.2591
7.0587	43.5102	53.4058	65.1119	72.8350

Table 4.1: Transformation errors

The 3D space mismatch between the robot end effector and our target object are also calculated by

$$\text{Mismatch} = \sqrt{x^2 + y^2 + z^2}. \tag{4.1}$$

With the table of errors in each direction and overall mismatch in 3D, the error distribution is analyzed and shown it in Figure 4.2.

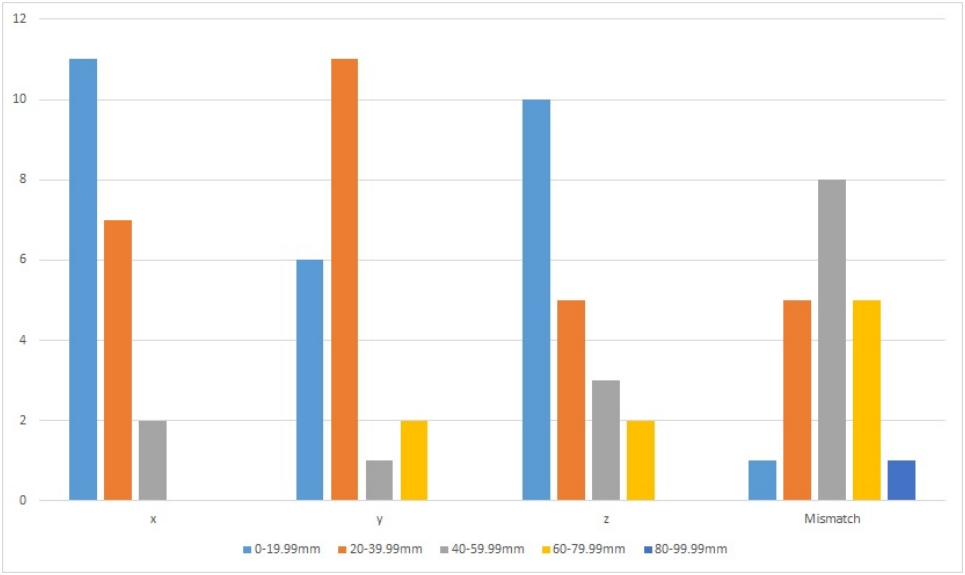


Figure 4.2: Error chart

We are able to tell from the chart that the error in each direction lies mostly under 40 mm and the overall 3D mismatch between robot end effector and our target object is around 50 mm. This means that our robot end effector will have no problem reaching objects with a radius of at least 50 mm under most circumstances. There is also still lots of room for us to improve the system accuracy with refined calibration methods in the future.

APPENDIX

In this appendix, we list all the programs we wrote for this prototype system and describe explicitly step by step to run it on our laboratory test machine. We list all the programs that should run for the system in Table 4.2.

Name	Directory	Description
roscore	Installed with ROS	Collaborate and collect programs that are prerequisites of a ROS-based system
driver.py	/home/crl/catkin_ws/src/nxc-100_driver/scripts/	Y-M HP3JC control program
stereopsis.cpp	/home/crl/catkin_ws/src/nxc-100_driver/src/	Stereo vision program
listener.py	/home/crl/catkin_ws/src/nxc-100_driver/scripts/	Message passing protocol that connects vision system and robotic system
test_move.py	/home/crl/catkin_ws/src/nxc-100_driver/scripts/	Move Y-M HP3JC back to original pose

Table 4.2: List of programs

All the steps needed to run the system are listed as follows:

- Run Spatial Vision software in Windows operating system and perform a complete stereo camera calibration following the instructions in the software;
- Switch to Ubuntu 14.04 operating system and open the Windows partition. Under directory /User/CRL/Documents/SpatialVision/data/, open "Cam_Mat_L.txt" to make the data on the Windows 7 partition accessible by our program;
- Run "roscore" in terminal;
- First, run "rosrun nxc100_driver driver.py" in a new terminal;
- Then in a new terminal, run "rosrun nxc100_driver stereopsis.cpp";
- In yet another terminal, run "rosrun nxc100_driver listener.py";

- Then carefully put objects in front of camera;
- To reset the system:
  - Press “reset” on the teach pedant;
  - Stop “rosrun nxc100_driver listerner.py”;
  - Run “rosrun nxc100_driver test_move.py” to reset robot pos.

BIBLIOGRAPHY

[1] Seth Hutchinson, Gregory D Hager, Peter Corke, et al. A tutorial on visual servo control. *Robotics and Automation, IEEE Transactions on*, 12(5):651–670, 1996.

[2] François Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *Robotics & Automation Magazine, IEEE*, 13(4):82–90, 2006.

[3] François Chaumette and Seth Hutchinson. Visual servo control, part ii: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, 2007.

[4] Yung Chen and Gérard Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724–2729. IEEE, 1991.

[5] John T Feddema, CS George Lee, and O Robert Mitchell. Model-based visual feedback control for a hand-eye coordinated robotic system. *Computer*, 25(8):21–31, 1992.

[6] Francois Chaumette. Potential problems of stability and convergence in image-based and position-based visual servoing. In *The confluence of vision and control*, pages 66–78. Springer, 1998.

[7] Wenyi Zhao and Nagaraj Nandhakumar. Effects of camera alignment errors on stereoscopic depth estimates. *Pattern Recognition*, 29(12):2115–2126, 1996.

[8] Gerda Kamberova and Ruzena Bajcsy. Sensor errors and the uncertainties in stereo reconstruction. In *Workshop on Empirical Evaluation Methods in Computer Vision, Santa Barbara, California*. Citeseer, 1998.

[9] Enrique Cervera, François Berry, and Philippe Martinet. Is 3d useful in stereo visual control? In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1630–1635. IEEE, 2002.

[10] Enrique Cervera, Angel P Del Pobil, François Berry, and Philippe Martinet. Improving image-based visual servoing with three-dimensional features. *The International Journal of Robotics Research*, 22(10-11):821–839, 2003.

[11] Farrokh Janabi-Sharifi, Lingfeng Deng, and William J Wilson. Comparison of basic visual servoing methods. *Mechatronics, IEEE/ASME Transactions on*, 16(5):967–983, 2011.

[12] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Reconstructing building interiors from images. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 80–87. IEEE, 2009.

[13] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(8):1362–1376, 2010.

[14] Ozgur Yilmaz and Fatih Karakus. Stereo and kinect fusion for continuous 3d reconstruction and visual odometry. In *Electronics, Computer and Computation (ICECCO), 2013 International Conference on*, pages 115–118. IEEE, 2013.

[15] Shuai Zhang, Chong Wang, and SC Chan. A new high resolution depth map estimation system using stereo vision and depth sensing device. In *Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on*, pages 49–53. IEEE, 2013.

[16] Yucheng Wang and Yunde Jia. A fusion framework of stereo vision and kinect for high-quality dense depth maps. In *Computer Vision-ACCV 2012 Workshops*, pages 109–120. Springer, 2013.

[17] Wei-Chen Chiu, Ulf Blanke, and Mario Fritz. Improving the kinect by cross-modal stereo. In *BMVC*, volume 1, page 3. Citeseer, 2011.

[18] Gowri Somanath, Sholom Cohen, Bob Price, and Chandra Kambhamettu. Stereo+ kinect for high resolution stereo correspondences. In *3D Vision-3DV 2013, 2013 International Conference on*, pages 9–16. IEEE, 2013.

[19] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[20] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[21] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[22] Inc. Yaskawa America. Hp3jc: Dimensions and general info. pdf file, 2007.

[23] John J. Craig. *Introduction to robotics - mechanics and control (2. ed.)*. Addison-Wesley, 1989.

[24] Mark W. Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. *Robot modeling and control*. John Wiley & Sons, Hoboken (N.J.), 2006.

[25] Saeed B Niku. *Introduction to robotics: analysis, systems, applications*, volume 7. Prentice Hall New Jersey, 2001.

[26] Jack B Kuipers. *Quaternions and rotation sequences*, volume 66. Princeton university press Princeton, 1999.

[27] Mikko Kytö, Mikko Nuutinen, and Pirkko Oittinen. Method for measuring stereo camera depth accuracy based on stereoscopic vision. In *IS&T/SPIE Electronic Imaging*, pages 78640I–78640I. International Society for Optics and Photonics, 2011.