Autonomous Mobility and Manipulation of a 9-DoF WMRA

by

William Garrett Pence

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yu Sun, Ph.D.
Redwan Alqasemi, Ph.D.
Rajiv Dubey, Ph.D.
Srinivas Katkoori, Ph.D.

Date of Approval:
October 20, 2011

Keywords: Rehabilitation, Robotics, Mobile Manipulation, Visual Servoing, ADL

Dedication

This work is dedicated to my family, friends, colleagues, the women in my life that inspired me to continue on in my education, and especially to my beloved nephew Owen who lost his battle with leukemia yet inspired me with his strength, courage, and happiness.

Acknowledgements

I would like to thank all of my committee members for their support and contributions to this work. Dr. Redwan Alqasemi and Dr. Yu Sun were incredible mentors and contributed greatly to this work and the theory behind it. I especially thank Dr. Redwan Alqasemi and Dr. Rajiv Dubey for pioneering and implementing combined WMRA control that this work builds on. I would like to thank Dr. Rajiv Dubey for his mentoring, contributions, and financial support. Dr. Srinivas Katkoori was my inspiration for pursuing research and is an excellent professor.

I also thank all the members of the Center for Assistive, Rehabilitation and Robotics Technologies for their contributions to the project, and especially Paul Mitzlaff for handling much of the hardware and electrical portions of the projects, and Fabian Farelo for collaboration on dual-trajectory WMRA control.

Table of Contents

List of Figures

Abstract

The wheelchair-mounted robotic arm (WMRA) is a 9-degree of freedom (DoF) assistive system that consists of a 2-DoF modified commercial power wheelchair and a custom 7-DoF robotic arm. Kinematics and control methodology for the 9-DoF system that combine mobility and manipulation have been previously developed and implemented. This combined control allows the wheelchair and robotic arm to follow a single trajectory based on weighted optimizations. However, for the execution of activities of daily living (ADL) in the real-world environment, modified control techniques have been implemented.

In order to execute macro ADL tasks, such as a "go to and pick up" task, this work has implemented several control algorithms on the WMRA system. Visual servoing based on template matching and feature extraction allows the mobile platform to approach the desired goal object. Feature extraction based on scale-invariant feature transform (SIFT) gives the system object detection capabilities to recommend actions to the user and to orient the arm to grasp the goal

object using visual servoing. Finally, a collision avoidance system is implemented to detect and avoid obstacles when the wheelchair platform is moving towards the goal object. These implementations allow the WMRA system to operate autonomously from the beginning of the task where the user selects the goal object, all the way to the end of the task where the task has been fully completed.

Chapter 1 Introduction

1.1 Motivation

According to the 2010 US Census Bureau report on disabilities, about ten percent of the working-age population has a disability, and there exists a great disparity among the employment-to-population ratio for citizens with disabilities (1). Assistive arms have proven to be effective devices for users with disabilities. These robotic arms can assist users in workplace environments to greatly improve capabilities for populations with disabilities in the workforce. They can also be used as assistive devices throughout users' daily lives to improve their independence. Several commercial robotic arms have been developed specifically for assistive purposes, and can also be mounted on wheelchairs, such as the iARM and JACO (2).

Even though WMRAs reduce dependence on caregivers, teleoperation of the robotic arm and coordination between the wheelchair and robotic arm operations still prove to be difficult for many users. For users that are completely locked-in, such as in many cases of amyotrophic lateral sclerosis (ALS), users are unable to

practically teleoperate the robotic arm using a brain-computer interface (BCI) (3). For these reasons, it becomes desirable to design a WMRA system that executes complete ADL tasks from beginning to end with minimal user input using both the wheelchair motion and robotic arm manipulation. A WMRA system that can execute complete macro ADL tasks could greatly improve the independence of users with disabilities without the great cognitive burden of teleoperation.

In order to allow for fully autonomous mobility and manipulation in the real-world environment, several control algorithms must be implemented on the WMRA system. A graphical user interface (GUI) will first present the user with a live view from the eye-in-hand camera mounted on the end effector of the robotic arm. After the user selects the goal object, the WMRA system must approach this object while avoiding possible obstacles in its path. This is done with visual servoing using template matching and feature extraction (4). A collision avoidance algorithm keeps track of obstacles and avoids them if necessary. Once the goal object has been approached, high-resolution feature extraction is executed for the purposes of object detection and grasping.

1.2 Goals

Control methods for the complete 9-DoF WMRA system combining mobility and manipulation have previously been

implemented (3). These control systems will be introduced in the next chapter since this work builds on the existing control methodology. The main goal of this work is to use sensory data to implement control algorithms that allow autonomous execution of complete ADL tasks. Another method of control for the WMRA is by using an image-based visual servoing (IBVS) technique described in (2). Visual servoing is more desirable for the physical implementation since it is robust against dynamic moving environments and can overcome imprecisions of the hardware. We can use an IBVS visual servoing technique along with a monocular eye in hand camera mounted on the end effector to provide autonomous mobility and manipulation throughout the execution of ADL tasks. The input to the visual servoing system is the goal object selected by the user as well as the vision data, and the output is a set of velocities to control WMRA motion using Cartesian control. Other systems have demonstrated that visual servoing can be a reliable form of control for a 6-DoF assistive robotic arm as in (3) and (4). However, these implementations have their shortcomings as neither uses a physical WMRA system with combined mobility and manipulation, and neither implement a true 3-dimensional IBVS approach.

Although it may be intuitive to use this visual servoing system from beginning to end, there are some pitfalls to the physical

implementation. The method of feature extraction we use is very robust, but in cluttered environments where the goal object is far away, its reliability is very low due to great noise in the image. Since our system can use the wheelchair platform to approach objects very far away, it is possible that goal objects may be too far away for feature extraction to provide reliable data. Therefore, the implementations of this work can be split into two main sections that deal with two phases of the task execution: approaching the goal object and grasping the goal object. The flow from the approach phase to the grasp phase is controlled using weighted optimization to change the motion from strictly wheelchair motion (at the beginning of the task) to strictly arm motion (at the end of the task). This weighted optimization will be discussed in detail in the WMRA Control.

After the user has selected the object in the camera view, template matching and feature extraction are used to keep track of where the object is in the environment and to allow for visual servoing. The WMRA platform keeps track of the goal object and moves towards it. During approach, mostly the mobile wheelchair platform is moving while the robotic arm is moving very little. The system also must be able to detect obstacles in the path to the goal object and navigate around them autonomously, if possible. If the system cannot autonomously navigate around the obstacles, the user

is prompted to move the wheelchair in an assisted teleportation mode. The approach algorithm fuses visual servoing and potential fields collision avoidance techniques. At the end of the approach phase, the WMRA is close enough to the goal object such that the robotic arm can reach and grasp the goal object.

During grasping, the robotic arm autonomously orients itself to match the grasping orientation for the particular object, and then grasps the goal object with the gripper assembly mounted on the end effector. Using the eye-in-hand camera, the goal object is recognized using feature extraction and a set of objects in a database. Feature extraction allows the system to recognize the type of object as well as the grasping orientation. An image-based visual servoing technique is used to position and orient the manipulator. At the end of the grasping phase, the task is completed and the goal object can be delivered to the user on the wheelchair. Figure 1.1 visualizes the control flow implemented in this work.

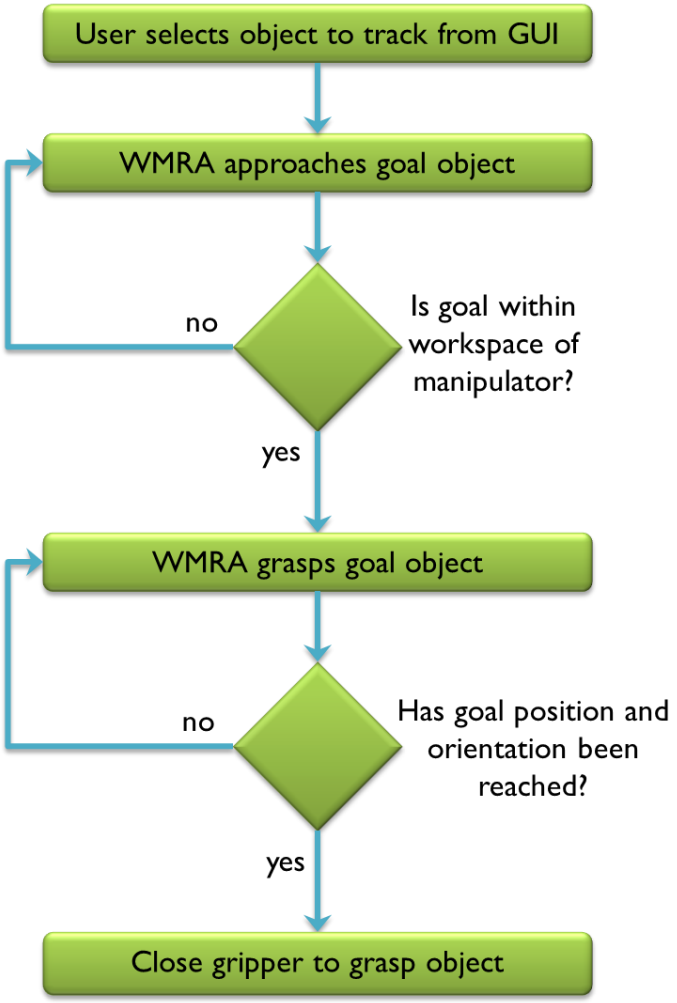

Figure 1.1: Control flow of this work

Finally, this work will present the physical testing results of these implementations on the WMRA during real-world ADL tasks. Several ADL tasks will be tested and motion and accuracy results will be presented. The next chapters will go into details on the background of the system and control algorithms implemented, and then the testing results will be presented along with a discussion and conclusion.

Chapter 2 Background

There is a great deal of opportunity for assistive robotics to increase independence of users with disabilities. Robotic devices of all kinds have helped users with disabilities to become more capable in the workplace as well as decrease their dependence on caregivers. Assistive robots can help to decrease the disparity between employment among persons with disabilities and persons without disabilities. In this section, we will outline several assistive robotic devices that have been previously developed and implemented. We will also discuss mobile manipulators, in which some form of assistive robotic arm is attached to a mobile platform. In addition, we will also discuss the control methodology for mobile manipulation as some problems arise in the control with mobile manipulators. We will describe the WMRA mobile manipulator in detail as this is the assistive system our work will deal with. Finally, we will cover the control of the 9-DoF WMRA system developed at the Center for Assistive, Rehabilitative, and Robotics Technologies (CARRT).

2.1 Assistive Robots

Today, the existence of robots in the world is commonplace. Robotics technologies have been used in various applications such as manufacturing, remote teleoperation, research, and many more. Arguably, the most important application of robotics deals with creating assistive devices that greatly improve the independence of humans with disabilities. Development of assistive robotics began with non-mobile workstation robots (8), which has led most recently to the development of lower-cost mobile devices that can be mounted in various places, or to a mobile platform.

Research with assistive robotics began with workstation robots, in which a robotic manipulator was permanently affixed to a workplace so that an operator could use the arm to execute tasks. The advantage to development of workstation robots is that they only need to be designed based on the set of tasks that are possible in its workplace location (8). Instead of creating a general-purpose robot, one could be built for a specific set of tasks inside the workplace. One example of a workstation robot was the Desktop Vocational Assistant Robot (DeVAR) developed at Stanford University (5). DeVAR used a commercial PUMA-260 robot mounted upside down on an adjustable track that allowed the arm to move back and forth in the workstation.

A gripper was customized that was suitable for its workplace-oriented tasks. Figure 2.1 shows the DeVAR system.

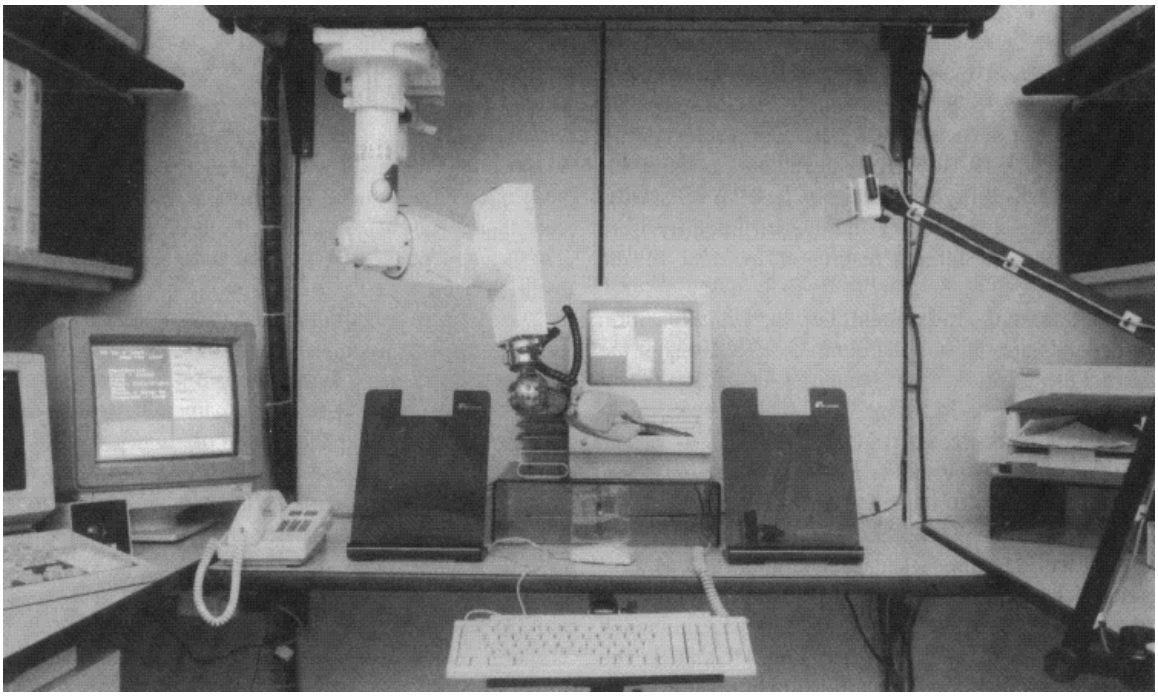

Figure 2.1: The DeVAR system developed at Stanford University (5)

Although workstation robots were effective for certain tasks inside a structured workplace, they were only useful at that specific location. For users that moved around to different locations, it is more desirable to have a smaller general-purpose robot that could be used in any location. Creating a mobile assistive robot that the user could carry with them would have a far greater impact on a users' independence.

2.2 Mobile Manipulators

Mobile manipulators can be defined simply as a robotic arm attached to a moving platform. These devices can be found in various

different fields ranging from space exploration to military surveillance. Mounting a robotic arm on a mobile platform greatly improves the workspace of the system by allowing the manipulator to reach any location that the mobile platform can travel to. As technology has improved, commercially-available robotic arms have become smaller and lighter, allowing them to be easily integrated on a wide array of mobile platforms.

Control of mobile manipulation has been studied extensively in research. The main advantage to mobile manipulators is that most of them inherently have redundancy (9), which allows them to be applied to several special-purpose applications. The kinematics of a 5-DoF manipulator and 2-DoF mobile platform have been described in (6) and allow a system that provides coordinated control to move the platform such that the target is within the workspace of the manipulator. This coordinated approach shows one control method for a redundant mobile manipulator. Another work described in (7) detailed combined kinematics for a non-holonomic mobile platform, such as a power wheelchair. Redundancy in the system was resolved using the projected gradient and reduced gradient optimization methods. In this work, a sample trajectory was followed where the manipulator was kept in a pre-specified orientation while the mobile

platform followed a circle. In these works, both the manipulator and mobile platform followed the same trajectory.

Other works have allowed for control systems that create separate trajectories of the manipulator and mobile platform. One example of this work is described in (8), where kinematic redundancies were resolved using separate controls for mobility and manipulation. By allowing separate trajectories for the manipulator and mobile platform, specialized tasks can be more easily executed. We continue describing coordinated mobility and manipulation control in a later section relating specifically to the WMRA system.

2.3 WMRAs

Many users with disabilities depend on a power wheelchair already, and mounting a portable robotic arm on that platform allows them to use the manipulator any place their wheelchair can go. A WMRA consists of a robot arm mounted on a mobile wheelchair platform. In the 1990s and early 2000s, two popular robotic arms were developed that could be mounted on a power wheelchair. The first of these was the Raptor robotic arm developed by Applied Resources (see Figure 2.2), which consists of a 4-DoF robotic arm and a planar gripper (9). The Manus manipulator (see Figure 2.3) is perhaps the more popular robotic arm, developed by Exact Dynamics (10). The Manus (or the iARM, which is a modified version of Manus) is

a 6-DoF robotic arm with a planar gripper. It was designed for Cartesian control using a joystick or keypad interface. The latest commercially-available robotic arm is the JACO (see Figure 2.4), developed by Kinova in 2009. The JACO consists of a 6-DoF robotic arm with a 3-finger gripper assembly. The main advantage to JACO is that it uses a 3-axis joystick interface, which makes teleoperation in Cartesian modes much easier than the Manus.

Figure 2.2: Applied Resources Raptor assistive arm (9)

Figure 2.3: Exact Dynamics iARM assistive arm (10)

Figure 2.4: Kinova JACO assistive arm (11)

These modern commercially-available robotic arms can be used for many general purpose applications, but are designed specifically to be used as a WMRA. Many persons with disabilities desiring an assistive robotic arm are already dependent on a power wheelchair (15), so it is intuitive to mount an appropriate manipulator onto their wheelchair platform so that they can use it throughout the course of

their daily lives. One important aspect of WMRA design is where to mount the manipulator such that it does not hinder the user's or wheelchair's movement and is able to be intuitively teleoperated (12). Through several research studies, WMRAs have proven to be effective assistive devices for users with disabilities.

Even though WMRAs have had a great impact on the independence of users with disabilities, their design and control can still be improved. Commercial manipulators such as the iARM and JACO consist of 6-DoF robotic arms, which are usually suitable for reaching a large workspace. However, expanding the design of the manipulator to a 7-DoF robotic arm allows for many optimizations through the redundant DoF. A 7-DoF system allows many different arm configurations while reaching the same end effector position and orientation. For the WMRA application, this is a very desirable feature for obstacle avoidance since the workspace of the robotic arm is confined to areas outside the wheelchair so that no joints come in contact with the user sitting on the wheelchair. Optimization of the redundant system allows the 7-DoF robotic arm to reach its desired positions more efficiently and based on the criterion than a 6-DoF robotic arm. Additionally, joint limit avoidance and singularity avoidance become more robust since a redundant DoF is available for manipulation.

The WMRAs developed at the Center for Assistive, Rehabilitation and Robotics Technologies consist of a 2-DoF power wheelchair and a 7-DoF robotic arm, providing for a complete 9-DoF system (see Figure 2.5). The robotic arm has 7 revolute joints with a gripper mounted on the end effector designed for generic ADL tasks. The power wheelchair is a standard wheelchair that is commercially available and has been modified. The advantage to the CARRT WMRA systems is mainly the added performance of the 7-DoF manipulator and combined control of both mobility and manipulation.

Figure 2.5: WMRA developed by CARRT at USF

Interface devices for WMRAs have traditionally been cumbersome for users and have a very large learning curve (15). Common interface devices for commercially-available WMRAs consist mainly of joysticks, keypads, eye gaze, voice recognition, and sip and puff devices. The large learning curve exists since it is not intuitive for humans to teleoperate a robotic arm in 3-dimensions using a 2-

dimensional interface device. Integrating 3-dimensional joysticks as with the JACO makes the device easier to teleoperate, but there still exists a learning curve for users to become practically efficient with the device. In order to improve control of WMRA systems beyond that of teleoperation using 3-dimensional input devices, it becomes necessary that the system become more task-oriented by adding autonomous capabilities to it.

Even though WMRA devices are mounted on the wheelchair platform, their control systems are still separated. In order to move the mobile wheelchair platform, the user must operate it with its dedicated joystick interface device, and then to manipulate the WMRA, the user must switch to the manipulator's interface device. In order to simplify the control systems, it becomes desirable to integrate both the wheelchair and WMRA control through a single interface device. To build a robust task-oriented control system, it also becomes desirable to coordinate control of both the wheelchair and robotic arm. To implement a robust task-oriented WMRA platform, control must take both mobility and manipulation into account.

2.4 WMRA Control[1]

The visual servoing control system developed in this work requires coordinated control of both mobility and manipulation. For the 7-DoF manipulator, numerical solutions exist to have it follow a desired trajectory (13). The other 2-DoF in the WMRA system are provided by the nonholonomic power wheelchair. The 2-DoF consist of linear translation and rotation about a fixed axis. When controlling the mobile platform, velocities must be given for the linear translation as well as rotation. We use the weighted least-norm solution with singularity-robust pseudo inverse to resolve redundancies in the mobile manipulator system. As we will discuss later, we also use this weighted optimization to control coordination of the wheelchair platform and robotic arm during executed ADL tasks. Combination of the robotic arm and wheelchair kinematics is done using Jacobian augmentation, which can give the flexibility of using conventional control and optimization methods without compromising the total coordinated control. Full kinematics and detailed equations can be found in a previous work concentrating on the control system (13).

[1] WMRA control theory is produced from (13)

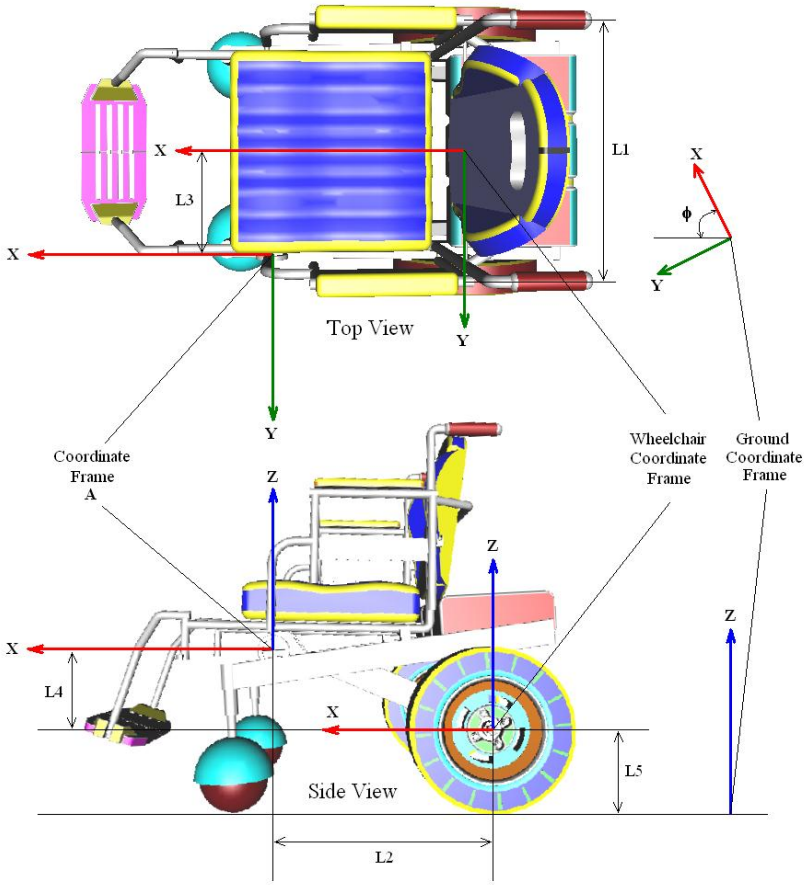

Figure 2.6: Coordinate frames of the WMRA (13)

Assuming that the manipulator is mounted on the wheelchair with L2 and L3 offset distances from the center of the differential drive across the x and y coordinates respectively, and L1 is the distance between the wheels (see Figure 2.6 for L-distances), then the mapping of the wheels' velocities to the manipulator's end effector velocity along its coordinates is defined by:

$$\dot{r}_c = J_c \times J_W \times \dot{\Theta}_c \qquad (2.1)$$

where J_c and J_w are the Jacobian matrices that map the arm base velocities to the end-effector velocities (without arm motion) and the

wheels' velocities to the arm base velocities, respectively. The wheelchair induced end effector velocity $\dot{r}_c$ and wheelchair velocity $\dot{\Theta}_c$ are:

$$\dot{r}_c = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\alpha} \quad \dot{\beta} \quad \dot{\gamma}]^T \tag{2.2}$$

$$\dot{\Theta}_c = \begin{bmatrix} \dot{\theta}_l \\ \dot{\theta}_r \end{bmatrix} \tag{2.3}$$

with:

$$J_c = \begin{bmatrix} [I]_{2x2} & \begin{bmatrix} -(P_{xg} \times S\emptyset + P_{yg} \times C\emptyset) \\ P_{xg} \times C\emptyset - P_{yg} \times S\emptyset \end{bmatrix} \\ [O]_{2x2} & [O]_{3x1} \\ [O]_{2x2} & 1 \end{bmatrix}_{6x3} \tag{2.4}$$

$$J_W = \frac{L_5}{2} \begin{bmatrix} C\emptyset + \frac{2}{L_1}(L_2 S\emptyset + L_3 C\emptyset) & C\emptyset - \frac{2}{L_1}(L_2 S\emptyset + L_3 C\emptyset) \\ S\emptyset - \frac{2}{L_1}(L_2 C\emptyset - L_3 S\emptyset) & S\emptyset + \frac{2}{L_1}(L_2 C\emptyset - L_3 S\emptyset) \\ \frac{-2}{L_1} & \frac{2}{L_1} \end{bmatrix} \tag{2.5}$$

where P_{xg} and P_{yg} are the x-y coordinates of the end-effector relative to the arm base frame, $\emptyset$ is the angle of the arm base frame (which is the same as the rotation angle of the wheelchair base), and L5 is the wheels' radius (see Figure 2.6). The above Jacobian and the Jacobian of the arm are combined together to control the end-effector.

The wheelchair will move forward when both wheels have the same speed and direction while rotational motion will be created when both wheels rotate at the same velocity but in opposite directions. Since the wheelchair's position and orientation are our control

variables rather than the left and right wheels' velocities, a relationship between the wheels' rotational velocities and the linear and rotational velocities of the wheelchair was derived $(\dot{X}, \dot{\emptyset})$:

$$\begin{bmatrix} \dot{\theta}_l \\ \dot{\theta}_r \end{bmatrix} = \begin{bmatrix} \frac{1}{L_5} & \frac{-L_1}{2 \cdot L_5} \\ \frac{1}{L_5} & \frac{L_1}{2 \cdot L_5} \end{bmatrix} \cdot \begin{bmatrix} \dot{X} \\ \dot{\emptyset} \end{bmatrix} \tag{2.6}$$

7-DoFs are provided by the robotic arm mounted on the wheelchair. From the DH parameters of the robotic arm specified in an earlier publication (13), the 6x7 Jacobian that relates the joint rates to the Cartesian speeds of the end effector based on the base frame is generated according to Craig's notation (14):

$$\dot{r}_A = J_A \cdot \dot{\Theta}_A \tag{2.7}$$

where $\dot{r}_A = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\alpha} \quad \dot{\beta} \quad \dot{\gamma}]^T$ is the task vector, $\dot{\Theta}_A = [\dot{\theta}_1 \quad \dot{\theta}_2 \quad \dot{\theta}_3 \quad \dot{\theta}_4 \quad \dot{\theta}_5 \quad \dot{\theta}_6 \quad \dot{\theta}_7]^T$ is the joint rate vector, and J_A is the robotic arm's Jacobian. By combining the wheelchair and arm kinematics using Jacobian augmentation, we find the total system kinematics (13).

Redundancy is resolved in the algorithm using weighted S-R inverse of the Jacobian to give a better approximation around singularities, and to use the optimization for different subtasks. Manipulability measure (15) is used as a factor to measure how far the current configuration is from singularity. This measure is defined as

$w = \sqrt{\det(J * J^T)}$. The S-R Inverse of the Jacobian in this case is defined as:

$$J^* = J^T * (J * J^T + k * I_6)^{-1} \tag{2.8}$$

where I_6 is a 6x6 identity matrix and *k* is a scale factor. It has been known that this method reduces the joint velocities near singularities, but compromises the accuracy of the solution by increasing the joint velocities error. Choosing the scale factor *k* is critical to minimize the error. Since the point in using this factor is to give approximate solution near and at singularities, an adaptive scale factor is updated at every time step to put the proper factor as needed:

$$k = \begin{cases} k_0 * \left(1 - \frac{w}{w_0}\right)^2 & \text{for } w < w_0 \\ 0 \text{ for } w \geq w_0 \end{cases} \tag{2.9}$$

where w_0 is the manipulability measure at the start of the boundary chosen when singularity is approached, and k_0 is the scale factor at singularity.

Weighted Least Norm solution proposed by (16) can be integrated to the control algorithm to optimize for secondary tasks. In order to put a motion preference of one joint rather than the other (such as the wheelchair wheels and the arm joints), a weighted norm of the joint velocity vector can be defined as:

$$|V|_W = \sqrt{V^T W V} \tag{2.10}$$

where *W* is a 9x9 symmetric and positive definite weighting matrix, and for simplicity, it can be a diagonal matrix that represent the motion preference of each joint of the system. For the purpose of analysis, the following transformations are introduced:

$$J_W = JW^{-1/2} \tag{2.11}$$

$$V_W = W^{-1/2}V \tag{2.12}$$

Using (2.8), (2.10), (2.11), and (2.12), it can be shown that the weighted least norm solution integrated to the S-R inverse is:

$$|V|_W = W^{-1}J^T(JW^{-1}J^T + k * I_6)^{-1}\dot{r} \tag{2.13}$$

The above method has been used in the 9-DoF WMRA system with the nine control variables (V) that represent the seven joint velocities of the arm and the linear and angular wheelchair's velocities. An optimization of criteria functions can be accomplished when used in the weighting matrix *W*.

The criteria functions used in the weight matrix for optimization can be defined based on different requirements. For the robotic arm, the physical joint limits can be avoided by minimizing an objective function that represents this criterion. One of these mathematical representations was proposed by (16) as follows:

$$H(q) = \sum_{i=1}^{7} \frac{1}{4} \cdot \frac{(q_{i,max} - q_{i,min})^2}{(q_{i,max} - q_{i,current}) \cdot (q_{i,current} - q_{i,min})} \tag{2.14}$$

where q_i is the angle of joint i. This criterion function becomes 1 when

the current joint angle is in the middle of its range, and it becomes infinity when the joint reaches either of its limits. The gradient projection of the criterion function can be defined as:

$$\frac{\partial H(q)}{\partial q_i} = \frac{(q_{i,max} - q_{i,min})^2 \cdot (2 \cdot q_{i,current} - q_{i,max} - q_{i,min})}{4 \cdot (q_{i,max} - q_{i,current})^2 \cdot (q_{i,current} - q_{i,min})^2} \tag{2.15}$$

When any particular joint is in the middle of the joint range, (2.15) becomes zero for that joint, and when it is at its limit, (2.15) becomes infinity, which means that the joint will carry an infinite weight that makes it impossible to move any further.

The diagonal weight matrix *W* can be constructed as:

$$W = \begin{bmatrix} w_1 + \left|\frac{\partial H(q)}{\partial q_1}\right| & 0 & \dots & \dots & 0 \\ 0 & w_2 + \left|\frac{\partial H(q)}{\partial q_2}\right| & 0 & \dots & 0 \\ \vdots & 0 & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & w_x & 0 \\ 0 & 0 & \dots & 0 & w_\varphi \end{bmatrix} \tag{2.16}$$

where w_i is a user-set preference value for each joint and w_x and w_φ are the weights associated with the position and orientation of the wheelchair. These values can achieve the user preference if joint limits are not approached and wheelchair motion is at its desired position.

We will later define criteria functions for the user-set preference values of the joints of the manipulator as well as those for the wheelchair. This weighted optimization using the weight matrix *W* allows us to coordinate mobility and manipulation during all stages of the autonomous task execution.

2.5 Vision-Based Control of Mobile Manipulators

Vision-based control has become popular in both fixed-base manipulators as well as mobile manipulators. The advantages of vision-based control become more prevalent in physical implementations of robotic systems where dynamic environments and inaccurate hardware are experienced. Vision-based control strategies, such as visual servoing, allow a system to approach and grasp objects by using a goal image saved in a database. This image is matched with the object in the camera image using some form of feature extraction, and the robot is manipulated until the camera image matches the goal image. Since this control strategy relies on live visual feedback information rather than strictly position-based control, it is able to overcome hardware inaccuracies such as slipping joints on a robotic arm or encoder position errors. Vision-based control is also robust against moving objects in a dynamic and cluttered environment since the control uses live feedback from the scene.

Image-based visual servoing (IBVS) is perhaps the most popular and simplest form of visual servoing. It provides a correspondence between matched features in the camera image and goal image and gives as output a velocity controller for the robot system. Therefore its control strategy is strictly based on image features rather than world positions. The features used in IBVS are immediately available in the images. Position-based visual servoing (PBVS) is another visual servoing control strategy in which 3D position of the goal object is estimated using various different methods. In this work, we concentrate on the IBVS technique. We will cover the mathematics behind the IBVS algorithm later on in Chapter 4. Visual servoing approaches are also defined in great detail in works such as (17) and (18), in which visual servoing in this work is based on.

Several works demonstrate an application of visual servoing in fixed-base as well as mobile manipulators. In (19), the Manus robotic arm was controlled using a visual servoing technique relying on color-based feature extraction. This implementation was fairly reliable at being able to grasp objects in an unstructured environment, but problems arose when objects with poor color information were selected. Rather than relying strictly on color information for tracking the goal object, in (3) the work was improved by using scale-invariant feature transform (SIFT) to track features between the goal and

camera image. This allowed a very robust visual servoing algorithm working towards autonomous grasping. The downside to this implementation is that the image of the goal object's desired pose must be saved in a database such that the environment must be somewhat structured. In a separate project (4), the Manus arm was used along with SIFT and a 2 1/2D visual servoing technique to autonomously grasp objects. This work split the motion into gross and fine motion, with different control systems for each phase. This approach did not implement a true 3D IBVS technique, but allowed objects to be grasped autonomously.

The aforementioned works concentrated on a fixed-base manipulator, so the workspace was limited to what the robotic arm could reach. A visual servoing technique extended to a mobile manipulator can greatly increase the workspace of the system, but also adds complexities concerning coordinated control of mobility and manipulation, collision avoidance for obstacles in the environment, and the possibility of losing the features being tracked due errant to movement of the mobile platform. There exist some works dealing with visual servoing of mobile platforms, but they typically involve very simple systems with low DoF (9). A more robust work that implements IBVS on a nonholonomic mobile manipulator with a 5-DoF robotic arm (20) also uses Q-learning to aid the mobile platform from

losing track of the visual features. This work decouples control of mobility and manipulation such that the mobile platform moves until the goal object is within the workspace of the manipulator, and then the manipulator grasps the object.

While these implementations prove that visual servoing is a robust and reliable control technique for fixed-base and mobile manipulators, they all have their shortcomings. Although some of the works provide an end to end autonomous solution for grasping objects (7), they do not use a true 3D IBVS technique. The works concentrating on fixed-base manipulators using the Manus arm can only grasp objects near the fixed-base. Expanding this work to a mobile manipulator such as a WMRA can greatly increase the abilities of the system. Works dealing with visual servoing of mobile manipulators use very simple robotic arms. Using a 7-DoF manipulator on the mobile platform would greatly increase the performance and capability of the entire system. Previous works focus on decoupling control of mobility and manipulation, but by coordinating these controls, the system can become much more stable and less choppy.

2.6 Visual Servoing of the 9-DoF WMRA

In this work, we desire to implement full 3D IBVS on the 9-DoF WMRA introduced above. To address the shortcomings of other works, we develop a control system that controls combined mobility and

manipulation simultaneously throughout the task. In order to design a reliable visual servoing control for the physical WMRA, we split the task into two phases.

During the approach phase, we use visual servoing with a single tracked point based on camshift (21), which gives us 2D velocity control initially. At the beginning of the approach phase, mostly wheelchair motion is used with limited arm motion. As the WMRA approaches the goal object, wheelchair motion should decrease as arm motion increases. Once a threshold distance from the end effector to the goal object is reached, we instantly switch to 3D IBVS used during the grasping phase.

By the time we reach the grasping phase and begin using 3D IBVS, the wheelchair system has slowed to a stop and the arm motion becomes entirely unrestricted. We use SIFT to extract and match features between the camera and goal image. Using SIFT with IBVS, and at least three matched points, we extract velocity control for full 6-DoF control based on the end effector of the WMRA. The arm positions and orients with respect to the IBVS velocity control until the velocities reach zero. At this point, the desired position and orientation has been reached and the system can now grasp the goal object. The gripper paddles are then closed to grasp the goal object, and it is

delivered via pre-programmed position control. The task has now been completed using both approach and grasping phases.

Chapter 3 Approach

During the approach phase, the WMRA system uses combined mobility and manipulation to approach the goal object such that it can be grasped. The goal object is selected by the user through the GUI screen and is tracked using methods described below. Motion is controlled using weighted optimization, and the criteria functions based on the image data are defined in the following sections. A potential fields collision avoidance method is also implemented during the approach phase to avoid possible obstacles detected using proximity sensors. At the end of the approach phase, the system will be in a position and orientation to be able to grasp the goal object since it has been tracked throughout the phase.

3.1 Camshift Tracking

Since we are splitting up the autonomous task into approach and grasping phases, we can simplify the approach phase. Since mainly gross motion is required during this phase, it is not necessary to orient the manipulator during approach. We can use strictly 2-dimensional visual servoing to center the mobile platform and manipulator on the

goal object as it approaches while controlling coordination of mobility and manipulation using weighted optimization introduced in Section 2.4 above.

At the beginning of the approach phase, the user is presented with a live camera feed of the workspace. They select the goal object through the GUI by selecting that area of the camera image. Since we therefore have a selection of the area of the scene we need to approach, we can use a simple camshift technique implemented in the OpenCV open source computer vision library (21). Our camshift function returns the centroid of the matched object in the scene image. This single centroid point is used for 2-dimensional visual servoing as described in the following section.

3.2 Visual Servoing

For the approach phase, we use a method similar to visual servoing, but since mostly wheelchair motion is being utilized, it is only necessary for 2-dimensional visual servoing. In order to center on the selected area, we must adjust wheelchair motion so that the object's centroid reaches the center of the image plane, denoted by a=(c_u,c_v). Wheelchair motion is controlled through w_x and w_φ from (2.16), which control wheelchair translation and rotation about its fixed axis, respectively. Since we wish to initially use mostly wheelchair motion during this phase, we set w_1 through w_7 using a criteria function based

on distance so that the manipulator will move more as the WMRA system approaches the goal object.

The wheelchair translation w_x is mainly related to the distance from the camera frame to the goal object, in the camera frame's z-direction. We can approximate this distance by means of proximity sensor or disparity map generated from a stereoscopic camera mounted on the end effector. Since w_x is directly proportionate to the distance on z, we have:

$$w_x = \frac{z_i}{\lambda z} \tag{3.1}$$

where λ is an appropriate gain, z is the approximated distance from the camera frame to the goal object, and z_i is the initial distance from the camera frame to the goal object.

The desired wheelchair rotation w_φ is directly related to the 2-dimensional visual servoing error. Since setting w_φ is only able to minimize the error in the camera frame's x-direction, we compute the error $e(t)_x$ using:

$$e(t)_x = s_x - c_u \tag{3.2}$$

where s_x is the current location of the centroid of the matched template relating to the x-direction, and c_u is the desired location of the template which is the center of the image plane. Since w_φ is directly proportionate to $e(t)_x$ computed in (3.2), we have:

$$w_{\varphi} = \frac{e(t)_{x\,max}}{\lambda e(t)_x} \tag{3.3}$$

where λ is an appropriate gain and $e(t)_{x\,max}$ is the maximum possible error in the x-direction.

We also desire to set the user-set preference values for w_1 through w_7 in order to control arm motion. We should use mostly wheelchair motion when the goal object is far away, and use mostly arm motion when the goal is very close. Therefore, we define the arm's user-set preference values for all 7 joints from (2.16) as:

$$w_1 = w_2 = \cdots = w_7 = \lambda z \tag{3.4}$$

where λ is an appropriate gain and z is the approximated distance from the camera to the goal object. When the distance is high, we have a large weight for arm motion so that very little arm motion is allowed. When the distance is low, we have a small weight for arm motion so that full arm motion is allowed.

Using equations (3.1), (3.3), and (3.4) we can set wheelchair motion so that the WMRA will approach the selected goal object area. As the wheelchair approaches the goal object, translational velocity resulting from w_x will decrease until it reaches zero, while the rotational velocity will be manipulated such that the WMRA centers on the goal object. Once the WMRA system has approached a predefined distance from the goal object such that the object is within the

workspace of the robotic arm, the grasping phase begins as described in Chapter 4.

3.3 Potential Fields

The WMRA system has been designed to be a modular platform where proximity sensors of various kinds can be mounted in several different orientations. In order to give physical distance information for our collision avoidance, we use simple infrared proximity sensors mounted on the forward part of the mobile platform. Since mostly forward motion is used in our visual servoing autonomous task execution, we are mainly only concerned with obstacles that may exist in the forward direction of the WMRA.

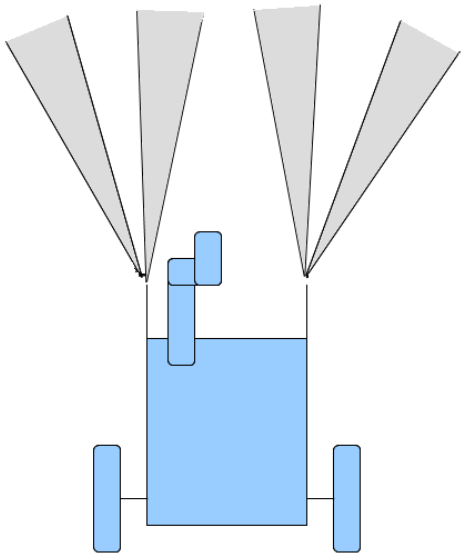

Figure 3.1: Proximity sensors mounted on the WMRA

Figure 3.1 shows the positions and range cones of the four infrared proximity sensors mounted on the WMRA mobile platform. We use Sharp GP2Y0A21YK sensors mounted on brackets.

Figure 3.2: Stereoscopic camera on the WMRA

In addition to the infrared proximity sensors, we can also use a stereoscopic camera to create a disparity map. A Point Grey Research BumbleBee 2 camera is mounted on the end effector, as seen in Figure 3.2. We use Point Grey's API to extract a disparity map. Similar to the physical sensors, we group the disparity map into zones. We then compute the average intensity values, or average distance, for each zone in the disparity map. These intensities are calibrated with the physical sensors with respect to distance of obstacles. Figure 3.3 shows a sample disparity map with the zone areas noted.

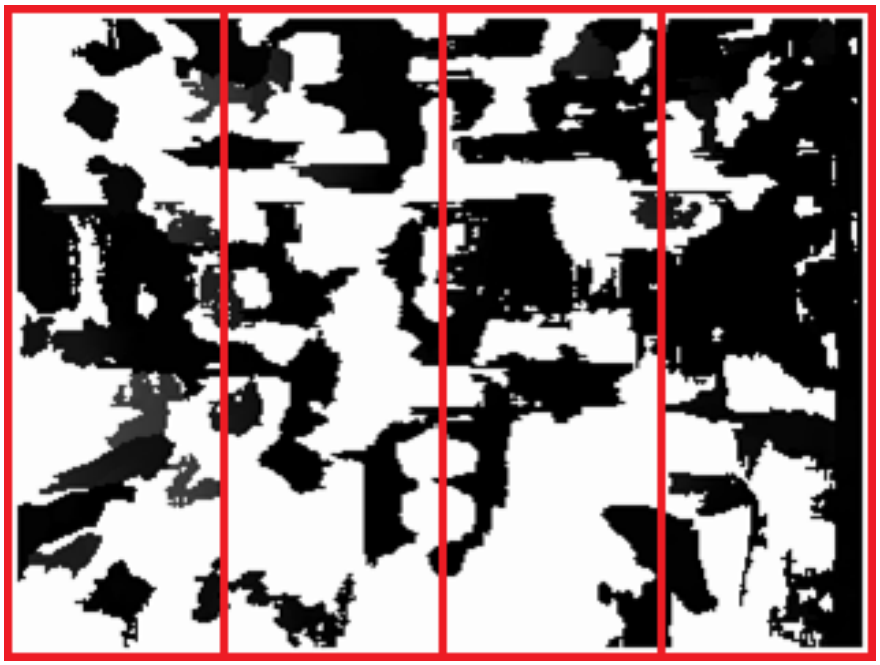

Figure 3: Disparity map and zone areas

Fusing both physical and computer vision sensors allow the collision avoidance system to be much more reliable. Obstacles that may not be recognized using stereoscopic vision are picked up by the physical sensors. With the addition of stereoscopic vision, we can use computer vision to estimate positions of objects in parts of the control algorithms in the future. We use a simple potential fields method using the physical distances measured by the infrared proximity sensors. This provides a vector value that can be used along with our visual servoing weights computed above.

3.4 Fusing Visual Servoing and Potential Fields

We can fuse the data we receive from our visual servoing and potential fields systems. We take the attractive force from the visual

servoing since this is the direction the system should travel based on the image data. We take the repulsive force from the potential fields collision avoidance since this is the direction the system should avoid due to collision with a detected obstacle.

From the sensor positions shown in Figure 3.1 above, we see that there are eight zones. For each zone, we combine the attractive and repulsive forces. We modify w_φ from (3.3) so that it is computed for each zone:

$$w_{\varphi_i} = \frac{e(t)_{x\,max}}{\lambda e(t)_{x_i}} - \vec{r_i} \tag{3.5}$$

where $\vec{r_i}$ is the repulsive force from the proximity sensor distance for zone *i* and $e(t)_{x_i}$ relates to the attractive force from the visual servoing system. The value w_{φ_i} is computed for each sensor zone, and then the control system chooses the w_{φ_i} with the greatest value and moves in that direction. This system allows the WMRA to detect obstacles using the proximity sensor array and then navigate around the obstacle to continue approaching the goal object. If the goal object leaves the camera frame, then the system halts and the user is prompted to teleoperate and then reselect the goal object.

3.5 Task Execution

At the beginning of the autonomous ADL task execution, the user is first presented with a GUI screen where a view of the

workspace is displayed through the eye-in-hand monocular camera mounted on the end effector. The user selects the desired goal object by clicking on a part of the object on the screen. As described in Section 3.1, we use the camshift algorithm developed in the OpenCV open-source computer vision library. Figure 3.4 shows the GUI before and after selecting the goal object. The user is provided with feedback by means of the camshift program drawing a red circle around the tracked object.

Figure 3.4: GUI for approach phase

If at any time the camshift algorithm fails, the entire system halts and prompts the user to reselect the goal object on the same GUI. This is important because during rare cases, the camshift algorithm may return an errant centroid that would cause large velocities for mobility or manipulation on the WMRA system. Code has

been implemented to detect an errant centroid in camshift, and the user is prompted to reselect the goal object after the system immediately halts.

Since the weights controlling arm motion are controlled based on the distance from the camera to the goal object, initially the arm moves very little and mostly the wheelchair platform moves. The platform centers in the x-direction as it approaches in the z-direction. These movements are computed based on (3.1) and (3.3), while arm motion is computed based on (3.4). When the system has almost approached the goal object, wheelchair movement is minimized until it halts while arm motion has increased to full motion. Once the system reaches a threshold distance from the goal object, the grasping phase begins as described in Chapter 4.

Chapter 4 Grasping

At the end of the approach phase, the WMRA is positioned so that strictly arm motion can be used to orient and position the manipulator to grasp the goal object. At this point, the WMRA is close enough so that the camera can see good detail of the goal object. We can now use a feature extraction method since we are close enough to the goal object. As long as we have at least three matched keypoints, we are able to use a full 3-dimensional visual servoing technique to position and orient the manipulator. At the end of the grasping phase, the gripper is positioned so that when the paddles are closed, the goal object is grasped. The grasped goal object can then be delivered to the user sitting in the wheelchair by means of pre-programmed position control where the gripper is positioned so that the user can reach and take the goal object.

4.1 SIFT Feature Extraction

It should be noted that any feature extraction method can be used with visual servoing control. However, since the reliability of the velocity control output by the IBVS system depends on the reliability

of the extracted features, a reliable and accurate algorithm should be used. SIFT (2) was developed by David Lowe by combining several image processing techniques. The algorithm extracts feature vectors from the image that are invariant to translation, size, rotation, illumination, and geometric distortion. A k-d tree algorithm is used to index these extracted features and to remove false matches. Features are clustered using Hough transforms, and the clusters are verified using a linear least squares method. Finally, based on a probabilistic model outliers can be rejected. Lowe's SIFT feature extraction and matching algorithms have proven to be very robust, especially due to its invariance to image transformations and differences typical in real-world image processing. The downside to the SIFT algorithm is that performance is very low due to intensive processing required.

Lowe's SIFT implementation has been provided to the community by means of a closed-source binary executable. Rob Hess provided an open-source implementation of SIFT using the OpenCV open-source computer vision library in (22). Hess's open-source implementation provided the same performance and results of Lowe's original closed-source implementation. In our program, we use parts of Hess's open-source SIFT implementation.

4.2 Image-Based Visual Servoing

Visual servoing relies on sets of features extracted from a goal image and a scene image and then compares them to compute the velocities needed to match the scene image with the goal image. The goal image is a sample image taken from the eye in hand camera when the end effector has reached its desired position and orientation. Sample goal and scene images can be viewed in Figure 4.1. Velocities outputted from the IBVS move the WMRA system until it has reached the goal orientation. At this point, the gripper paddles can close and grasp the goal object, and the task is completed.

Figure 4.1: Sample scene (left) and goal (right) image

We desire to have a reliable and accurate method of feature extraction since the reliability of the visual servoing control relies on accurate feature extraction. We use the SIFT algorithm as described in Section 4.1 above. SIFT performance is improved on the WMRA system by saving the set of features extracted from the goal image so

that it is not searched at every iteration. Performance is further improved by reducing the resolution slightly, and only searching areas in the scene image that are likely to contain goal image features. For our code implementation, we use the open source SIFT library developed by Rob Hess (22).

The goal of visual servoing is to minimize an error computed by:

$$e(t) = s(m(t), a) - s^* \tag{4.1}$$

where the features extracted in the scene image that match features from the goal image are represented by *s(m(t),a)*, where *m(t)* is the vector of image measurements and *a* is a set of camera parameters. In our case, *m(t)* consists of the image coordinates of the matched features in the scene image. From this point forward, we can represent *s(m(t),a)* simply as *s*. The vector s^* consists of the desired goal image measurements. In our case, s^* contains the image coordinates of the features in the goal image. Therefore, from (4.1), we see that the error *e(t)* is simply the difference between *s* and s^*.

For our application, we desire to design a velocity controller that can control the WMRA system using this visual servoing in Cartesian control based on the end effector. The relationship between the time variation of *s* and the camera velocity is described by:

$$\dot{s} = L_s v_c \tag{4.2}$$

where L_s is the image Jacobian related to *s*. The term image Jacobian is used interchangeably with feature Jacobian and interaction matrix. The vector v_c is the velocity controller for the WMRA system, which consists of v_c and ω_c, the instantaneous linear velocity and angular velocity, respectively, in all three dimensions. For visual servo control, $v_c=(v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$. Using (4.1) and (4.2), we find the relationship between the time variation of the error and the camera velocity:

$$\dot{e}=L_e v_c \tag{4.3}$$

where $L_e=L_s$. We wish to solve (4.3) for v_c so that we can use it as velocity input to the WMRA control system. Therefore, we finally find:

$$v_c=-\lambda L_e^{+} e \tag{4.4}$$

where λ is a gain for the velocity control and the Moore-Penrose pseudo-inverse of L_e is taken to solve for v_c.

We now define the image Jacobian to use in (4.4). We must first relate the 3-dimensional point X=(*X*,*Y*,*Z*) to the 2-dimensional point x=(*x*,*y*):

$$\begin{aligned} x &= \frac{X}{Z} = (u - c_u) \\ y &= \frac{Y}{Z} = (v - c_v) \end{aligned} \tag{4.5}$$

where m=(*u*,*v*) from (4.1) above is the coordinates in pixels of the image feature point, and a=(c_u,c_v) is the set of camera parameters with the principal point described by c_u and c_v. The image Jacobian is a

6x2*k* matrix for *k* matched feature points. The image Jacobian L_x, related to x from (4.5) is:

$$L_x = \begin{bmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix} \tag{4.6}$$

where *Z* is the estimated distance of the feature point from the camera frame and *x* and *y* are from (4.5). In order to control the WMRA system using 6-DoF Cartesian control, we must have at least *k*=3 matched feature points to determine the velocities. We stack the interaction matrices for *k* points:

$$L_x = \begin{bmatrix} L_{x_1} \\ L_{x_2} \\ \vdots \\ L_{x_k} \end{bmatrix} \tag{4.7}$$

Similarly, we also stack the errors such that e from (4.4) is:

$$e = \begin{bmatrix} e_{x_1} \\ e_{y_1} \\ e_{x_2} \\ e_{y_2} \\ \vdots \\ e_{x_k} \\ e_{y_k} \end{bmatrix} \tag{4.8}$$

We have now designed a visual servoing control system based on (4.4) from (17) and (18) that can output velocity control for the WMRA so that the system can minimize the error such that a selected goal object can be approached for execution of ADL tasks. When the visual error has been minimized and the velocities of the system

approach zero, then the robotic arm has reached its desired position and orientation. At this time, the gripper paddles can be closed to grasp the goal object and deliver it to the user in the wheelchair.

4.3 Task Execution

We switch from the approach phase to the grasping phase when a threshold distance on *z* between the camera frame and goal object is reached. This switch is immediate and seamless so that the user sitting in the wheelchair does not experience any disruption in wheelchair or arm movement. No further input from the user is required during the grasping phase such that the entire execution of the ADL task is autonomous from beginning to end. Feedback is given to the user by means of a GUI based on Hess's open-source SIFT implementation (see Figure 4.2).

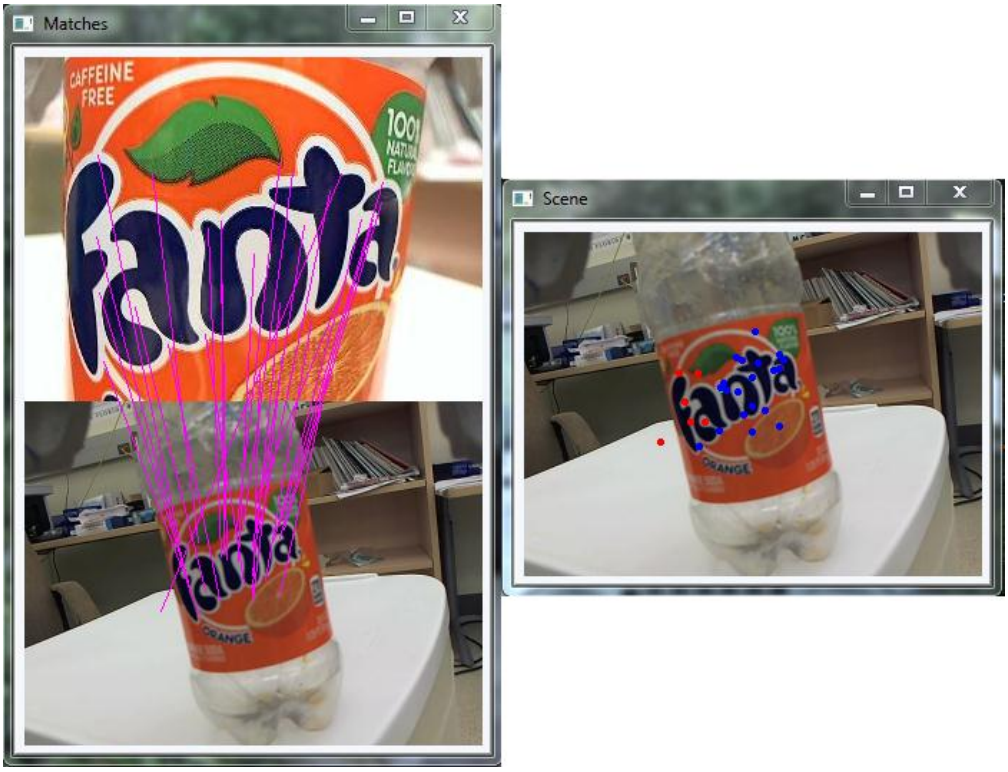

Figure 4.2: GUI for grasping phase

From Figure 4.2, we can see that matched features are visualized on the left. We run several noise reduction algorithms inside the SIFT code to reduce the number of false SIFT feature matches. The screen on the right shows positive matched SIFT features in blue and rejected false matched SIFT features in red. Once the goal position and orientation has been reached as determined strictly by image data, the gripper paddles close to grasp the goal object and it is delivered to the user sitting in the wheelchair. We will examine data and results from physical testing of these task executions in Chapter 5 below.

Chapter 5 Physical Testing of ADL Tasks

In order to demonstrate the physical results of the 9-DoF combined visual servoing theory described above, we design an ADL task that the system can autonomously execute and provide data and results below. Physical design of the WMRA can be reviewed in Section 2.3 and 2.4 above as well as in (13) in further detail. Figure 5.1 shows the 9-DoF WMRA platform used for physical testing in this work.

Figure 5.1: The 9-DoF WMRA system used for testing

The gripper assembly has been slightly modified in order to mount an eye-in-hand monocular camera for visual servoing. We use a standard commercially-available USB webcam for the eye-in-hand camera, specifically a Logitech C910. For estimating the distance between the camera frame and the goal object, we use an infrared proximity sensor. The Sharp GP2Y0A21YK proximity sensor is mounted directly beneath the camera. Figure 5.2 shows the camera and proximity sensor mounted beneath the gripper assembly.

Figure 5.2: Camera and proximity sensor on gripper assembly

5.1 Description of ADL Tasks

To demonstrate an application of this 9-DoF visual servoing combined mobility and manipulation, we design an ADL task that can be executed autonomously from beginning to end using this system.

We use a "go to and pick up" ADL task where the user selects the goal object on the GUI and the 9-DoF WMRA system approaches and then grasps the goal object autonomously. For this task, we place a goal object far away from the WMRA system so that movement of the mobile platform is necessary to successfully grasp the goal object. This demonstrates combined mobility and manipulation of our control system. The WMRA uses the wheelchair and arm to center on the goal object and approach it. When a threshold distance from the camera frame to the goal object is reached, the grasping phase then begins and the manipulator is positioned and oriented to grasp the goal object. Finally, the gripper paddles close to grasp the goal object and it is delivered to the user sitting in the wheelchair.

During teleoperation of the WMRA system for this "go to and pick up" ADL task, the user would first use the joystick to move the wheelchair close enough such that the goal object is within the workspace of the robotic arm. The user would then switch to controlling the arm by means of various user interfaces provided, such as laptop touch screen control. After the gripper is correctly positioned and oriented, a command would be sent to close the gripper paddles. Finally, a command would be sent to move the arm back to a position in reach of the user for them to retrieve the goal object.

During autonomous execution of this "go to and pick up" ADL task, the only user input would be initially selecting the goal object on the GUI screen. After the object was selected, the approach phase would begin where combined mobility and manipulation are used to move the WMRA close to the goal object while centering with the wheelchair and arm. When the WMRA is close enough, the grasping phase will begin and strictly arm motion will position and orient so that the goal object is within the paddles of the gripper. At this time, the gripper closes the paddles and delivers the goal object to the user sitting in the wheelchair so that they can retrieve it.

5.2 Physical Testing Results

We execute the "go to and pick up" task autonomously with several different objects. Each object is enrolled in the image database for visual servoing so a positive match exists for the goal image of that particular object. Sample results from the physical execution of the approach phase can be seen in Figure 5.3.

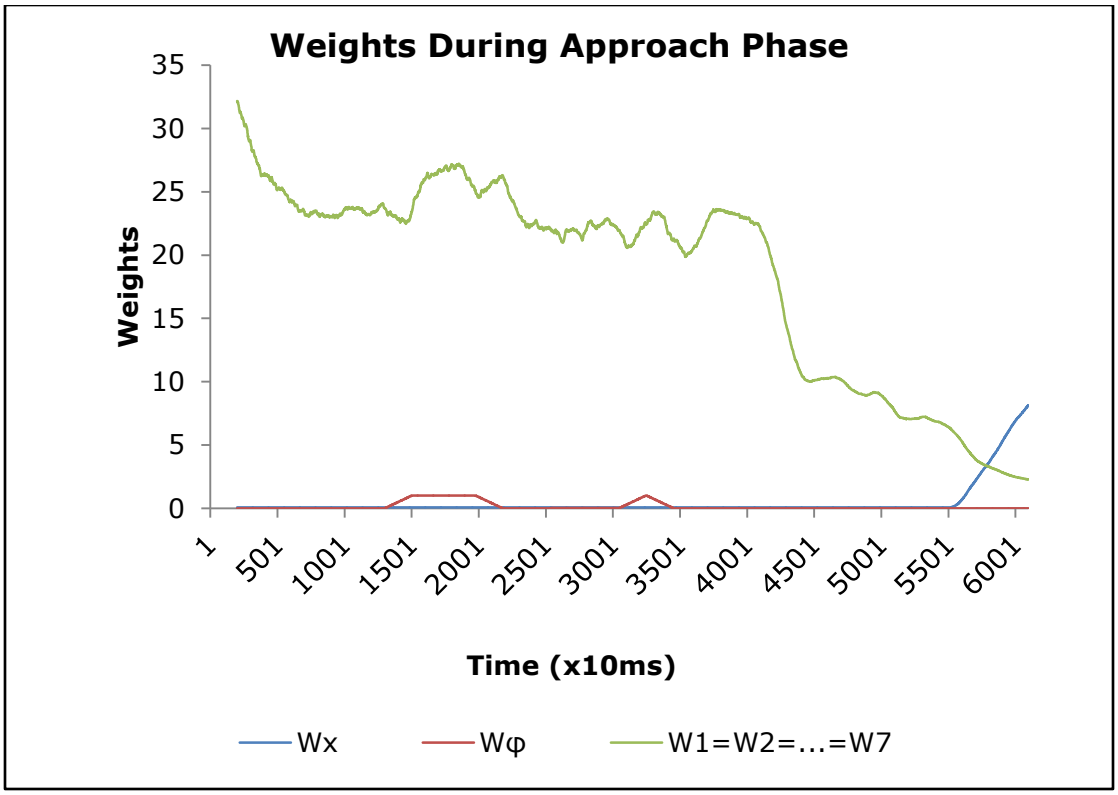

Figure 5.3: Weights during the approach phase

As we can see, initially very little arm motion is used, where the arm weight $w_1=w_2=\ldots=w_7$ is very high. As the system approaches the goal object and the distance on z is reduced, the resulting arm weight reduces until it becomes very low and full arm motion is used. Since initially platform motion should be used mostly, we see that w_x is low. As the distance on z is reduced, w_x becomes very large once it approaches the goal object. In this manner, the wheelchair motion is reduced until it halts during the switch to the grasping phase. Rotational movement of the wheelchair is controlled with w_φ where the weight depends on the necessary rotational movement to center the wheelchair on the goal object during approach.

When the system switches to the grasping phase, the weights on the wheelchair are set to infinity so that no further mobility is used. Arm weight is minimized so that full manipulation is possible, except for when joint limits or singularities prevent movement. 3-dimensional IBVS is now used to position and orient the arm. The velocity output of the IBVS system can be visualized in Figure 5.4 and Figure 5.5.

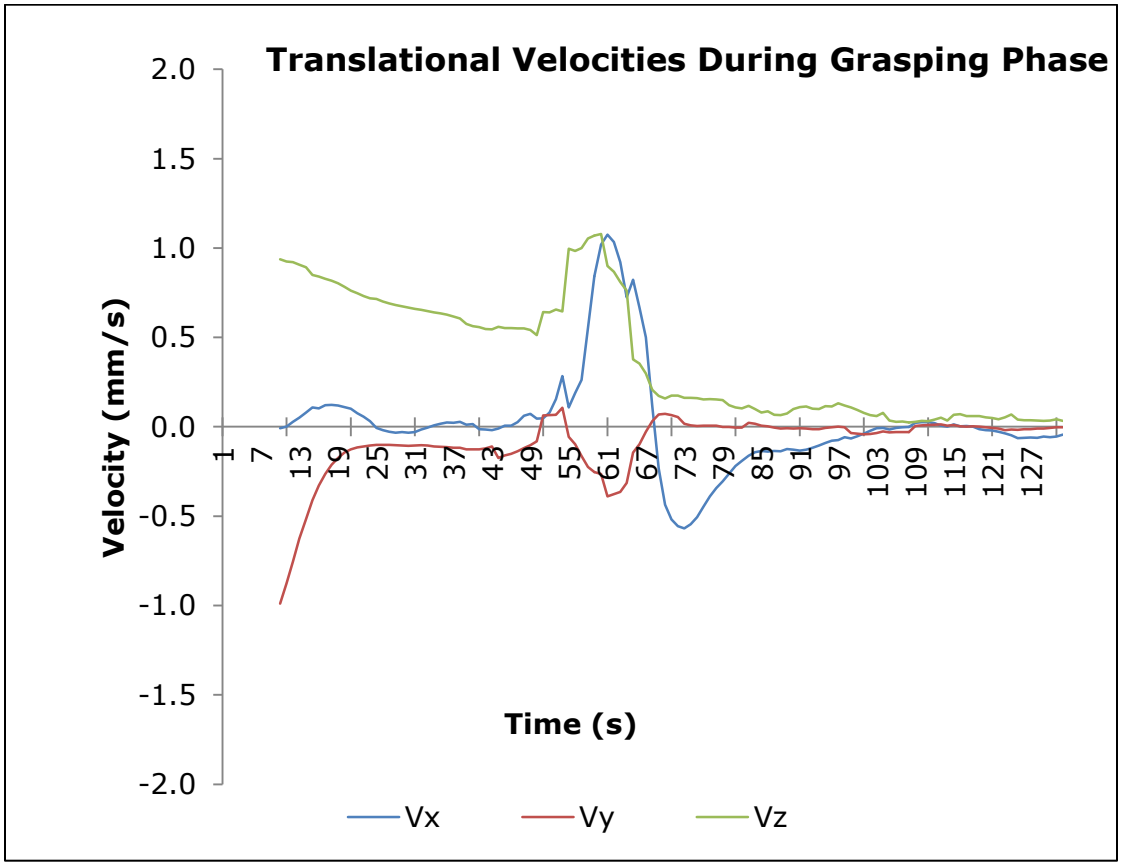

Figure 5.4: Translational velocities during the grasping phase

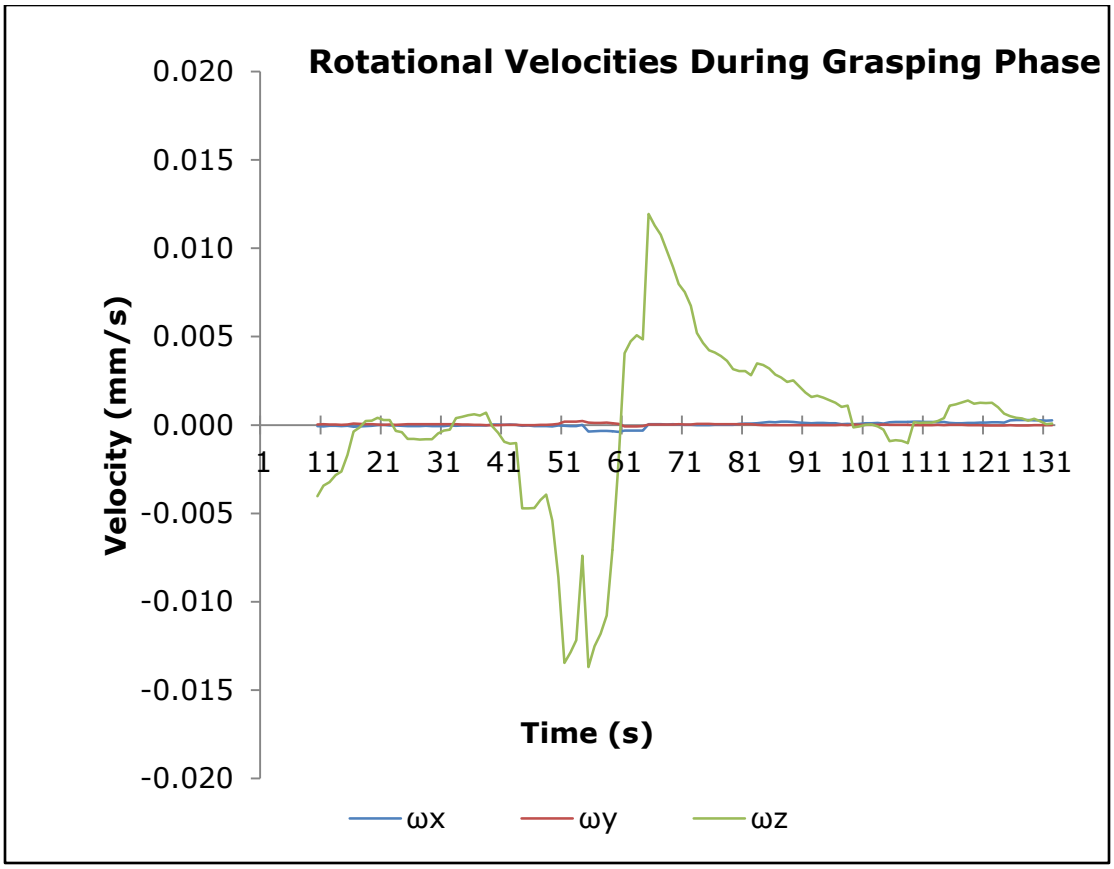

Figure 5.5: Rotational velocities during the grasping phase

As we can see from Figure 5.4 and Figure 5.5, the velocities for the end effector converge at a minimum at the end of the grasping phase. Although some noise exists in the IBVS velocity output, the system stays stabile during testing and is able to grasp the goal object (see Figure 5.6).

Figure 5.6: Grasping the goal object

Reliability of the physical system is generally very good, and typically the control system will end in a successful grasp. After testing the "go to and pick up" task 30 times with the same initial and goal positions, the system resulted in successful task execution 83.33% of the time. During rare cases where the goal object is lost during the approach phase, the entire system immediately halts and the user is prompted to reselect the goal object. In some cases when poor image features exist due to environmental effects such as lighting or cluttered backgrounds, the system experiences additional noise, but most of the time once the camera gets close enough to the goal object, good features can then be extracted and stability increases. Most failed executions were a result of less than desirable accuracy of the infrared proximity sensor on the end effector for estimating the

distance from the system to the goal object. Use of a more reliable sensing device would improve reliability of the system. Execution time for the "go to and pick up" task over 30 trials averaged 2 minutes and 16 seconds with a standard deviation of 47 seconds. Minimum execution time was 28 seconds and maximum was 3 minutes and 37 seconds. The variation in execution time depended on the amount of arm movement necessary during the grasping phase. Figure 5.7 shows the end of the task when the goal object has been delivered to the user sitting in the wheelchair.

Figure 5.7: WMRA at the end of the ADL task

Chapter 6 Discussion and Conclusion

6.1 Discussion

In this work, we have presented a control theory implementing visual servoing control on a 9-DoF mobile manipulator system for autonomous execution of ADL tasks. In this work, control of mobility and manipulation is combined and used simultaneously throughout the execution of ADL tasks. This provides a streamlined control system resulting in smooth and seamless physical operation for beginning to end autonomous execution of ADL tasks. During the final grasping phase, full 3-dimensional IBVS is used such that objects of virtually any position and orientation can be grasped.

The advantages to autonomous execution of the demonstrated "go to and pick up" task are fairly obvious. Teleoperated control of the 9-DoF WMRA system is difficult, even when used by able-bodied users. When users with reduced upper-body mobility teleoperate the system, this difficulty is greatly magnified. For extreme cases such as users that are locked in, the BCI must be used and teleoperation of the complex WMRA system results in a very great cognitive burden on the

user, and execution of the ADL task takes a very long period of time. By automating control of mobility and manipulation, macro tasks can be developed so that users only have to select an ADL activity they wish to execute.

6.2 Conclusion

The advantages to using vision-based control for the physical implementation of autonomous WMRA control are vast. By using visual servo control, inaccuracies of the hardware can be overcome. Vision-based control also does not require that workspaces be as structured as in position-based control. Visual servoing is also robust against dynamic obstacles as well as noisy and cluttered environments. The physical results and high success of grasping for the vision-based approaches implemented in this work show that it is a very strong implementation for autonomous execution of ADL tasks.

This work provides a control theory using full 3-dimensional IBVS implemented on a 9-DoF mobile manipulator. Mobility and manipulation are controlled in a combined manner such that they are used simultaneously throughout the control flow. This provides a very robust system that is streamlined and reliable for autonomous execution of macro ADL tasks.

6.3 Future Work

Although we have successfully executed a very simple "go to and pick up" task, many other ADL tasks can be executed using the vision-based control developed in this work for the 9-DoF WMRA system. In the future, macro tasks such as "go to and open the door" can be implemented using this work. A BCI interface (see Figure 6.1) is also being developed so that users can select macro tasks based on object recognition techniques. This would allow a user to select an area of the screen, and when the program detects the object they will be presented with a pool of ADL tasks to choose from.

Figure 6.1: Sample BCI and interface screen

Further research also involves doing human testing with both teleoperated and autonomous ADL tasks with the WMRA system. The

WMRA is unique in that there is always a human user on-board that the program can leverage knowledge from. Certain tasks are very difficult for computers to execute, such as object detection. However, humans can very easily detect objects with much greater accuracy. Human subject testing can help us to understand which parts of the ADL task are very difficult to teleoperate and should be automated, and which parts are very easy to teleoperate and should be done by the human user.

References

1. Bureau, US Census. *Disability Among the Working Age Population: 2008 and 2009.* s.l. : http://www.census.gov/prod/2010pubs/acsbr09-12.pdf, September 2010.

2. *Distinctive Image Features from Scale-Invariant Keypoints.* Lowe, David. s.l. : International Journal of Computer Vision 2004.

3. *Vision-based control of the Manus using SIFT.* Liefhebber, F. and Sijs, J. s.l. : IEEE 10th International Conference on Rehabilitation Robotics (ICORR 2007), 2007, pp. 854-861.

4. *Eye-in-hand stereo visual servoing of an assistive robot arm in unstructured environments.* Kim, Dae-Jin, Lovelett, Ryan and Behal, Aman. s.l. : IEEE International Conference on Robotics and Automation (ICRA 2009), May 2009, pp. 2326-2331.

5. *VA/Stanford Rehabilitation Robotics Research and Development Program: Lessons Learned in the Application of Robotics Technology to the Field of Rehabilitation.* Loos, H.F.M. Van der. s.l. : IEEE Transactions on Rehabilitation Engineering, March 1995, Vol. 3, pp. 46-55. 1.

6. *The Kinematics of a Redundant Mobile Manipulator.* C. Ding, P. Duan, M. Zhang and H. Liu. s.l. : Proceedings of the IEEE International Conference on Automation and Logistics, 2009.

7. *Kinematic Modeling and Redundancy Resolution for Nonholonomic Mobile Manipulators.* A. Luca, G. Oriolo, and P. Giordano. Orlando, FL : Proceedings of the 2006 IEEE International Conference on Robotics and Automation, 2006.

8. *Experimental Evaluation of Dynamic redundancy resolution in a Nonholonomic wheeled Mobile Manipulator.* G.D. White, R.M. Bhatt, C Pei Tang and V.N. Krovi. s.l. : IEE/ASME Transactions on Mechatronics, 2009, Vol. 14. 3.

9. *The Raptor wheelchair robot system.* Mahoney, R. M. Netherlands : IOS, 2001, Vol. Integration of Assistive Technology in the Information Age, pp. 135-141.

10. *Technical Results from Manus User Trials.* H.Eftring, K.Boschian. s.l. : Proceedings of the 1999 ICORR, 1999. pp. 136-141.

11. Kinova Jaco Arm - Robotnik. *Robotnik.* [Online] http://www.robotnik.es/en/products/robotic-arms/kinova-jaco-arm.

12. *Integrating robotic research: a survey of robotic wheelchair development.* Yanco, Holly A. Stanford, California : AAAI Spring Symposium on Integrating Robotic Research, March 1998.

13. *Maximizing Manipulation Capabilities for People with Disabilities Using a 9-DoF Wheelchair-Mounted Robotic Arm System.* Dubey, R. Alqasemi and R. s.l. : Proceedings of the 2007 IEEE 10th International Conference on Rehabilitation Robotics (ICORR 2007), 2007.

14. Craig, J. *Introduction to Robotics, Mechanics, and Control.* s.l. : Addison-Wesley Publishing, 2003. 0201543613.

15. Yoshikawa, T. *Foundations of Robotics: Analysis and Control.* s.l. : MIT Press, 1990. 0262240289.

16. *A Weighted Least-Norm Solution Based Scheme for Avoiding Joint Limits for Redundant Joint Manipulators.* Dubey, T. Chan and R. 2, s.l. : IEEE Robotics and Automation Transactions (R&A Transactions), 1995, Vol. 11, pp. 286-292.

17. *Visual servo control. I. Basic approaches.* Chaumette, F. and Hutchinson, S. 4, s.l. : Robotics & Automation Magazine, IEEE, 2006, Vol. 13, pp. 83-90.

18. *A tutorial on visual servo control.* Hutchinson, S., Hager, G.D. and Corke, P.I. 5, s.l. : Robotics and Automation, IEEE Transactions on, 1996, Vol. 12, pp. 651-670.

19. *Collaborative control of the MANUS manipulator.* Driessen, B., et al., et al. s.l. : 9th International Conference on Rehabilitation Robotics (ICORR 2005), June 2005, pp. 247-251.

20. *A Hybrid Visual Servo Controller for Robust Grasping by Wheeled Mobile Robots.* Wang, Ying, Lang, Haoxiang and de Silva, C.W. 5, s.l. : IEEE/ASME Transactions on Mechatronics, October 2010, Vol. 15, pp. 757-769.

21. *Computer video face tracking for use in perceptual user interface.* Bradski, G.R. s.l. : Intel Technology Journal, Q2, 1998.

22. *An open-source SIFT Library.* Hess, Rob. New York, NY : Proceedings of the international conference on Multimedia (MM '10), ACM, 2010, pp. 1493-1496.

Appendices

Appendix A Source Code

Main Application

```
#include <iostream>
#include <fstream>
#include <Afxwin.h>
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
#include "controlMotor.h"
#include "match.h"
#include "wmraLJ.h"
#include "main.h"
#include "camshift.h"
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>


#define dacm 15 //speed modifier for when batteries die down
#define pwmStop 130 //idle speed PWM value for wheelchair
#define Tmod 25 //gain to translation velocity control
#define wmod 25 //gain to rotational velocity control
#define tmaxv 10 //maximum translational velocity control
#define wgain 1 //gain for sending the weight to WMRA Opt()

int pwmX = 0, pwmY = 0; //wheelchair platform control
int wmraEnd = 1; //end flag for WMRA control program
extern float v;
extern int choice6; //go back to ready position when 1
extern int c; //flag for ending the camshift thread
extern int cc; //flag for communicating camshift errors
extern int track_object; //camshift, =0 no object tracked, =1 object
  tracked
CvCapture *capture = 0; //pointer to camera object
double centroidX = 0, centroidY = 0; //coordinates of center (2D)
double wmraCtrl[10] = {0, 0, 0, 0, 0, 0, 0, 1, 135, 135};
double armWeight = 1; //weight for the arm during approach
/*wmraCtrl[0] -> ARM forward (1)/backward(-1) Tz
wmraCtrl[1] -> ARM left(1)/right(-1) Tx
wmraCtrl[2] -> ARM up(1)/down(-1) Ty
wmraCtrl[3] -> ARM yaw (.003/-.003) wz
wmraCtrl[4] -> ARM roll (.003/-.003) wx
wmraCtrl[5] -> ARM pitch (.003/-.003) wy
wmraCtrl[6] -> ARM gripper open(-1)/close(1)
To STOP arm and stay idle, set wmraCtrl[0...6]=0
wmraCtrl[7] -> WMRA program exit(0)/run(1)
wmraCtrl[8] -> PLATFORM forward(idle++)/backward(idle--) (PWM, 55-215)
            135 idle, 135-165 forward, 105-135 backward
wmraCtrl[9] -> PLATFORM right(idle++)/left(idle--) (PWM, 55-215)
            135 idle, 135-165 right, 105-135 left
To STOP platform and stay idle, wmraCtrl[8]=wmraCtrl[9]=135
*/
```

```
char *tempChar; //temporary char pointer passed to thread

using namespace std;

UINT camshiftThread (LPVOID pParam)
{
   //thread for camshift process
   if (camshift()) //calls the camshift object tracking program
   {
      cerr << "There was a problem starting the camshift thread!" <<
        endl;
      return 1;
   }
   return 0;
}

UINT wmraThread (LPVOID pParam)
{
   //thread for moving WMRA
   wmraEnd = wmraControl(); //calls main WMRA program
   //AfxEndThread(0);
   return 0;
}

int main ()
{
   double prat[5] = {5,5,5,5,5};
   int xWeight=0, yWeight=0, xWeightI=0, count=0, flag=0, j=0;
   int nxWeight=0, nyWeight=0, nv=0;
   int numFeatures=0; //number of matched features
   double Z=2.5; //distance from camera frame to goal object
   double wphi=0, dacx=0;
   double Tx=0, Ty=0, Tz=0, wx=0, wy=0, wz=0; //velocity controls from
     visual servoing system
   IplImage *frame; //scene image
   IplImage *templ = cvLoadImage ("crush.jpg", 1); //template image from
     file
   int n1=0;
   double *px; //x-coordinates of the goal image
   double *py; //y-coordinates of the goal image
   double *nx; //x-coordinates of the scene image
   double *ny; //y-coordinates of the scene image
   double *xd; //differences in x-direction (for e)
   double *yd; //differences in y-direction (for e)
   double *xx; //differences in x-direction (for x in Lx)
   double *yy; //differences in y-direction (for y in Lx)
   int *nf; //pointer to convert number of matched features
   struct feature* feat1;
   double *stats;
   double **viserv;
   stats = (double *) malloc (3 * sizeof (double));
   stats[0] = 0;
   stats[1] = 0;
```

```
stats[2] = 0;
clock_t start, end;

//<------------------------ START CAMSHIFT ------------------------>
cout << "Please select an object to track by left-clicking on a part
  of the object in the video feed..." << endl;
AfxBeginThread (camshiftThread, tempChar);

//<------------------------ START LAB JACK ------------------------>
cout << "Initializing platform..." << endl;
if (Initialize())
{
  cerr << "There was an error initializing the Lab Jack!" << endl;
  return 1;
}

//set wheelchair to idle pwm initially
wmraCtrl[8] = pwmStop;
wmraCtrl[9] = pwmStop;

//<------------------------ START WMRA CODE ------------------------>
cout << "Initializing WMRA..." << endl;
AfxBeginThread (wmraThread, tempChar);
cout << "WMRA initialized..." << endl;

v = 25; //set initial WMRA arm speed

cout << "Platform initialized and idle motion set, is is now safe to
  turn on joystick..." << endl;
cout << "Joystick must be turned on within 10 seconds or before an
  object is selected, whichever is longer..." << endl;

Sleep(10000); //wait for everything to get settled, then start visual
  servoing

//check to see if object has been selected by user
if (track_object == 0)
{
  while (!track_object)
  {
  }
}

Sleep(1000);

//set up file for printing out weights
fstream weights("weights.csv", ios::out);
weights << "wx,wphi,warm" << endl; //print header

cout << "Object has been selected and is now being tracked..." <<
  endl;

start = clock ();
```

Appendix A (Continued)

```
//<------------------------ APPROACH OBJECT ------------------------>
//control loop for initial visual servoing (approach object)

//set initial xweight
xWeightI = abs (320-centroidX);

//move platform forward
wmraCtrl[8] = pwmStop + 10;

if (GetAIN(4, Z)) //read I1 proximity sensor
{
   cerr << "There was a problem reading I1 proximity sensor!" << endl;
   wmraCtrl[8] = pwmStop;
   wmraCtrl[9] = pwmStop;
   return 1;
}
Z=0.5/Z;
armWeight = Z * wgain; //update the weights W for arm motion in Opt()
if (Z < 1)
{
   dacx = Z;
}
else
{
   dacx = 1;
}

while (Z > 0.3) //while distance threshold not reached
{
   //compute velocity based on errors (distance from image center)
   xWeight = abs (320-centroidX);
   nxWeight = 320-centroidX;
   yWeight = abs (240-centroidY);
   nyWeight = 240-centroidY;

   if (Z < 0.7)
   {
      v = (max(xWeight, yWeight))/5; //pick max/1.5 for velocity of
         arm
   }
   else
   {
      v = (max(xWeight, yWeight))/1.5; //pick max/1.5 for velocity of
         arm
   }

   if (xWeight > yWeight)
   {
      nv = nxWeight / 1.5;
   }
   else
   {
      nv = nyWeight / 1.5;
   }
```

Appendix A (Continued)

```
//computing phi-weight for rotation of wheelchair
//if ((xWeight < 13) || (flag == 1))
if (xWeight < 13)
{
   wphi = 1;
   flag = 1;
}
else
{
   wphi = 0.000125 * ( (((xWeightI-xWeight)*(xWeightI-
      xWeight))*((xWeight+xWeightI)*(xWeight+xWeightI))) /
      ((xWeightI*xWeightI)*xWeight));
}

weights << dacm*dacx << ",";
weights << wphi << ",";

//if target has been lost, or is too small, then pause WMRA and
   prompt user to re-select target
if (cc < 0)
{
   wmraCtrl[8] = pwmStop;
   wmraCtrl[9] = pwmStop;
   wmraCtrl[0] = 0;
   wmraCtrl[1] = 0;
   wmraCtrl[2] = 0;
   wmraCtrl[3] = 0;
   wmraCtrl[4] = 0;
   wmraCtrl[5] = 0;

   track_object = 0;

   cout << "TARGET LOST, PLEASE RE-SELECT TARGET ON GUI!" << endl;

   while (!track_object)
   {
      //do foo
   }

   Sleep (1000);
}

if ((dacm*dacx)<10)
{
   wmraCtrl[8] = pwmStop + 10;
}
else
{
   wmraCtrl[8] = pwmStop + (dacm * dacx); //move forward
}

if (count < 1000) //move arm forward for a bit
{
```

Appendix A (Continued)

```
      wmraCtrl[0] = 1;
   }
   else //stop moving arm forward
   {
      wmraCtrl[0] = 0;
   }

   //modifying wheelchair movements for w-phi
   //if (centroidX<240)
   if (centroidX<270)
   {
      //move platform left
      wmraCtrl[9] = pwmStop - (((1-wphi)*dacm)*dacx);
   }
   //else if (centroidX>400)
   else if (centroidX>370)
   {
      //move platform right
      wmraCtrl[9] = pwmStop + (((1-wphi)*dacm)*dacx);
   }
   else //centered in x-direction
   {
      //arm idle in x-direction
      //wmraCtrl[9] = (1-wphi) = 0
      wmraCtrl[9] = pwmStop;
   }

   if (centroidX<300)
   {
      //move arm left
      wmraCtrl[1] = 1;
   }
   else if (centroidX>340)
   {
      //move arm right
      wmraCtrl[1] = -1;
   }
   else //centered in x-direction
   {
      //arm idle in x-direction
      wmraCtrl[1] = 0;
   }

   if (centroidY<220)
   {
      //move arm up
      wmraCtrl[2] = 1;
   }
   else if (centroidY>260)
   {
      //move arm down
      wmraCtrl[2] = -1;
   }
   else //centered in y-direction
```

```
    {
      //arm idle in y-direction
      wmraCtrl[2] = 0;
    }

    if (GetAIN(4, Z)) //read I1 proximity sensor
    {
      cerr << "There was a problem reading I1 proximity sensor!" <<
        endl;
      wmraCtrl[8] = pwmStop;
      wmraCtrl[9] = pwmStop;
      return 1;
    }
    Z=0.5/Z;
    armWeight = Z * wgain; //update the weights W for arm motion in
      Opt()
    weights << armWeight << endl; //print the arm weights
    if (Z < 1)
    {
      dacx = Z;
    }
    else
    {
      dacx = 1;
    }

    count++; //increment count for arm movement forward
  }

  v = 1;
  armWeight = 0;

  //set all motions back to idle
  wmraCtrl[0] = 0;
  wmraCtrl[1] = 0;
  wmraCtrl[2] = 0;
  wmraCtrl[3] = 0;
  wmraCtrl[4] = 0;
  wmraCtrl[5] = 0;
  wmraCtrl[8] = pwmStop;
  wmraCtrl[9] = pwmStop;

  c = 27; //end camshift thread

  cout << "System has now approached object." << endl;

  cout << "Please wait, re-initializing camera..." << endl;
  Sleep (2000);
  capture = cvCaptureFromCAM(0); //only 1 camera used, we pass 0
  if (!capture)
  {
    cerr << "There was an error opening camera. Program will
      terminate!" << endl;
    return 1;
```

```
}

cout << "Starting SIFT IBVS..." << endl;

//set up file for printing the velocity control
fstream velocity ("velocities.csv", ios::out);
velocity << "Tx,Ty,Tz,wx,wy,wz" << endl; //print header

//<------------------------ GRASP OBJECT ------------------------>
//control loop for initial visual servoing (approach object)

if (GetAIN(4, Z)) //read I1 proximity sensor
{
   cerr << "There was a problem reading I1 proximity sensor!" << endl;
   wmraCtrl[8] = pwmStop;
   wmraCtrl[9] = pwmStop;
   return 1;
}
Z=1/Z;

flag = 0;

//while (!flag) //while distance threshold not reached
while (Z > 0.41)
{
   if (!templ)
   {
      cerr << "There was an error getting the template image!" <<
        endl;
      return 1;
   }

   frame = cvRetrieveFrame (capture);

   if (!frame)
   {
      cerr << "There was an error getting the frame image!" << endl;
      return 1;
   }

   IplImage *framelow = cvCreateImage (cvSize (320, 240), frame-
     >depth, frame->nChannels);
   //convert frame to 320x240
   cvResize (frame, framelow, 1);

   //stats = siftMatch (templ, framelow, &feat1, &n1);
   viserv = siftMatch (templ, framelow, &feat1, &n1);

   //grab all the data from viserv for local access here
   px = viserv[0];
   py = viserv[1];
   nx = viserv[2];
   ny = viserv[3];
   xd = viserv[4];
```

```
yd = viserv[5];
xx = viserv[6];
yy = viserv[7];
nf = (int *) viserv[8]; //cast double* to int*

//save number of matched features as int
numFeatures = *nf;

cout << "number matched features: " << numFeatures << endl;

if (numFeatures > 2)
{
  CvMat *vc = cvCreateMat (6, 1, CV_32FC1); //velocity control
  CvMat *Le = cvCreateMat (numFeatures*2, 6, CV_32FC1); //image
    Jacobian
  CvMat *pLe = cvCreateMat (6, numFeatures*2, CV_32FC1); //pseudo-
    inverse of image Jacobian
  CvMat *e = cvCreateMat (numFeatures*2, 1, CV_32FC1); //error
    matrix

  j = 0; //j is additional counter for traversing pointers

  for (int i=0; i<2*numFeatures; i+=2)
  {
    //set e: error matrix (2*nf,1)
    cvmSet (e, i, 0, xd[j]);
    cvmSet (e, i+1, 0, yd[j]);

    //set Le: image Jacobian (2*nf,6)
    cvmSet (Le, i, 0, -1/Z);
    cvmSet (Le, i, 1, 0);
    cvmSet (Le, i, 2, xx[j]/Z);
    cvmSet (Le, i, 3, xx[j]*yy[j]);
    cvmSet (Le, i, 4, -(1+xx[j]*xx[j]));
    cvmSet (Le, i, 5, yy[j]);
    cvmSet (Le, i+1, 0, 0);
    cvmSet (Le, i+1, 1, -1/Z);
    cvmSet (Le, i+1, 2, yy[j]/Z);
    cvmSet (Le, i+1, 3, 1+yy[j]*yy[j]);
    cvmSet (Le, i+1, 4, -xx[j]*yy[j]);
    cvmSet (Le, i+1, 5, -xx[j]);

    j++; //increment j
  }

  //compute pseudo-inverse of Le
  cvInvert (Le, pLe, CV_SVD);

  //compute vc=pLe*e
  cvMatMul (pLe, e, vc);
  //get the velocity controller data
  Tx = cvmGet (vc, 0, 0);
  Ty = cvmGet (vc, 1, 0);
  Tz = cvmGet (vc, 2, 0);
```

Appendix A (Continued)

```
      wx = cvmGet (vc, 3, 0);
      wy = cvmGet (vc, 4, 0);
      wz = cvmGet (vc, 5, 0);
   }
   else //not enough features matched, set to idle motion
   {
      Tx = 0;
      Ty = 0;
      Tz = 0;
      wx = 0;
      wy = 0;
      wz = 0;
   }

   //modify velocity control using gains
   Tx = -Tx/Tmod;
   Ty = -Ty/Tmod;
   Tz = Tz;
   wx = -wx/wmod;
   wy = -wy/wmod;
   wz = wz/wmod;

   //check to see if translational velocity exceeds maximum
   if (abs(Tx) > tmaxv)
   {
      if (Tx < 0)
      {
         Tx = -tmaxv;
      }
      else
      {
         Tx = tmaxv;
      }
   }
   if (abs(Ty) > tmaxv)
   {
      if (Ty < 0)
      {
         Ty = -tmaxv;
      }
      else
      {
         Ty = tmaxv;
      }
   }
   if (abs(Tz) > tmaxv/2)
   {
      if (Tz < 0)
      {
         Tz = -tmaxv/2;
      }
      else
      {
         Tz = tmaxv/2;
```

```
    }
  }

  cout << "Tx: " << Tx << endl;
  cout << "Ty: " << Ty << endl;
  cout << "Tz: " << Tz << endl;
  cout << "wx: " << wx << endl;
  cout << "wy: " << wy << endl;
  cout << "wz: " << wz << endl;

  velocity << Tx << ",";
  velocity << Ty << ",";
  velocity << Tz << ",";
  velocity << wx << ",";
  velocity << wy << ",";
  velocity << wz << endl;

  //set motion for Tx
  wmraCtrl[1] = Tx;

  //set motion for Ty
  wmraCtrl[2] = Ty;

  //set motion for Tz
  wmraCtrl[0] = Tz;

  //set motion for wx
  wmraCtrl[4] = wx;

  //set motion for wy
  wmraCtrl[5] = wy;

  //set motion for wz
  wmraCtrl[3] = wz;

  //update previous ratios
  prat[0] = prat[1];
  prat[1] = prat[2];
  prat[2] = prat[3];
  prat[3] = prat[4];
  prat[4] = Tz;

  if ((abs(prat[0]) < .05) && (abs(prat[1]) < .05) && (abs(prat[2]) <
    .05) && (abs(prat[3]) < .05) && (abs(prat[4]) < .05))
  {
    cout << prat[0] << " " << prat[1] << " " << prat[2] << " " <<
      prat[3] << " " << prat[4] << endl;
    flag = 1;
  }

  if (GetAIN(4, Z)) //read I1 proximity sensor
  {
    cerr << "There was a problem reading I1 proximity sensor!" <<
      endl;
```

```
      wmraCtrl[8] = pwmStop;
      wmraCtrl[9] = pwmStop;
      return 1;
    }
    Z=1/Z;

    cvReleaseImage (&framelow);
  }

  weights.close();
  velocity.close();

  v = 5;

  //set all motions back to idle
  wmraCtrl[0] = 1;
  wmraCtrl[1] = 0;
  wmraCtrl[2] = 0;
  wmraCtrl[3] = 0;
  wmraCtrl[4] = 0;
  wmraCtrl[5] = 0;
  wmraCtrl[8] = pwmStop;
  wmraCtrl[9] = pwmStop;

  //reach forward some to ensure grasp
  Sleep (7000);

  end = clock ();

  cout << "Execution time is " << end - start << endl;

  //set all motions back to idle
  wmraCtrl[0] = 0;
  wmraCtrl[1] = 0;
  wmraCtrl[2] = 0;
  wmraCtrl[3] = 0;
  wmraCtrl[4] = 0;
  wmraCtrl[5] = 0;
  wmraCtrl[8] = pwmStop;
  wmraCtrl[9] = pwmStop;

  cout << endl << endl << "WARNING: JOYSTICK SHOULD BE TURNED OFF NOW!"
    << endl << endl;

  //close the gripper
  //wmraCtrl[6] = 1;
  //Sleep(8000); //close for 7 seconds
  //wmraCtrl[6] = 0;

  wmraCtrl[7] = 0; //stop WMRA arm motion

  //Sleep(5000);
  //choice6 = 1; //go back to ready position
```

Appendix A (Continued)

```
   while (wmraEnd != 0) //wait for wmra thread to finish
   {
      //loop until WMRA thread is finished
   }

   cvReleaseCapture(&capture); //safely release OpenCV webcam feed

   return 0;
}
```

Appendix A (Continued)

Camshift Tracking for Approach Phase Based on (21)

```
/* This file is based on the camshift demo program bundled with
the OpenCV 2.0 library and is based on the work in [21] */

#include <stdio.h>
#include <ctype.h>
#include <iostream>
#include <string>
#include "camshift.h"
#include "cv.h"
#include "highgui.h"

extern CvCapture *capture; //pointer to camera object
extern double centroidX, centroidY;
IplImage *image = 0, *hsv = 0, *hue = 0, *mask = 0, *backproject = 0,
*histimg = 0;
CvHistogram *hist = 0;
int select_object = 0;
int track_object = 0;
int show_hist = 1;
int c = 0;
int cc = 0;
CvPoint origin;
CvRect selection;
CvRect track_window;
CvBox2D track_box;
CvConnectedComp track_comp;
int hdims = 16;
float hranges_arr[] = {0,180};
float* hranges = hranges_arr;
int vmin = 10, vmax = 256, smin = 30;

using namespace std;

void on_mouse (int event, int x, int y, int flags, void* param)
{
    if( !image )
        return;

    if( image->origin )
        y = image->height - y;

    if( select_object )
    {
     selection.width = 5;
     selection.height = 5;
     select_object = 0;
     track_object = -1;
    }

    switch( event )
    {
    case CV_EVENT_LBUTTONDOWN:
```

```
            origin = cvPoint(x,y);
            selection = cvRect(x,y,0,0);
            select_object = 1;
            break;
        }
    }

    CvScalar hsv2rgb (float hue)
    {
        int rgb[3], p, sector;
        static const int sector_data[][3]=
            {{0,2,1}, {1,2,0}, {1,0,2}, {2,0,1}, {2,1,0}, {0,1,2}};
       hue *= 0.033333333333333333333333333333333f;
        sector = cvFloor(hue);
        p = cvRound(255*(hue - sector));
        p ^= sector & 1 ? 255 : 0;

        rgb[sector_data[sector][0]] = 255;
        rgb[sector_data[sector][1]] = 0;
        rgb[sector_data[sector][2]] = p;

        return cvScalar(rgb[2], rgb[1], rgb[0],0);
    }

    //=============================================================
    // camshift() is called by the main application. This
    // function initializes the camera and displays a video
    // feed. The user selects an object in the video display by
    // left-clicking and holding down while selecting the
    // object. The function then loops while updating the (x,y)
    // coordinates of the center. The coordinates (0,0) are
    // sent until the user selects an object in the window.
    // This information is used for the visual servoing in the
    // main application. Returns 1 for error condition.
    //=============================================================
    int camshift ()
    {
      double differenceX=0, differenceY=0, prevX=0, prevY=0;

      cout << "Please wait, initializing camera..." << endl;

        capture = cvCaptureFromCAM( 0 );

        if(!capture)
        {
            cerr << "Could not initialize capturing..." << endl;
            return 1;
        }

        cvNamedWindow( "CamShiftDemo", 1 );
        cvSetMouseCallback( "CamShiftDemo", on_mouse, 0 );

        for(;;)
        {
```

```
        IplImage* frame = 0;
        int i, bin_w;

        frame = cvQueryFrame( capture );
        if( !frame )
            break;

        if( !image )
        {
            /* allocate all the buffers */
            image = cvCreateImage( cvGetSize(frame), 8, 3 );
            image->origin = frame->origin;
            hsv = cvCreateImage( cvGetSize(frame), 8, 3 );
            hue = cvCreateImage( cvGetSize(frame), 8, 1 );
            mask = cvCreateImage( cvGetSize(frame), 8, 1 );
            backproject = cvCreateImage( cvGetSize(frame), 8, 1 );
            hist = cvCreateHist(1,&hdims,CV_HIST_ARRAY,&hranges,1);
            histimg = cvCreateImage( cvSize(320,200), 8, 3 );
            cvZero( histimg );
        }

        cvCopy( frame, image, 0 );
        cvCvtColor( image, hsv, CV_BGR2HSV );

        if( track_object )
       {
            int _vmin = vmin, _vmax = vmax;

            cvInRangeS( hsv, cvScalar(0,smin,MIN(_vmin,_vmax),0),
                        cvScalar(180,256,MAX(_vmin,_vmax),0), mask );
            cvSplit( hsv, hue, 0, 0, 0 );

            if( track_object < 0 )
            {
                float max_val = 0.f;
                cvSetImageROI( hue, selection );
                cvSetImageROI( mask, selection );
                cvCalcHist( &hue, hist, 0, mask );
                cvGetMinMaxHistValue( hist, 0, &max_val, 0, 0 );
                cvConvertScale( hist->bins, hist->bins, max_val ?
                   255. / max_val : 0., 0 );
                cvResetImageROI( hue );
                cvResetImageROI( mask );
                track_window = selection;
                track_object = 1;

                cvZero( histimg );
                bin_w = histimg->width / hdims;
                for( i = 0; i < hdims; i++ )
                {
                    int val = cvRound( cvGetReal1D(hist-
>bins,i)*histimg->height/255 );
                    CvScalar color = hsv2rgb(i*180.f/hdims);
```

```
                    cvRectangle( histimg, cvPoint(i*bin_w,histimg-
>height),
                                 cvPoint((i+1)*bin_w,histimg->height
- val),
                                 color, -1, 8, 0 );
                }
            }

            cvCalcBackProject( &hue, backproject, hist );
            cvAnd( backproject, mask, backproject, 0 );
            cvCamShift( backproject, track_window,
                        cvTermCriteria( CV_TERMCRIT_EPS |
CV_TERMCRIT_ITER, 10, 1 ),
                        &track_comp, &track_box );
            track_window = track_comp.rect;

            if( !image->origin )
                track_box.angle = -track_box.angle;
            cvEllipseBox( image, track_box, CV_RGB(255,0,0), 3,
CV_AA, 0 );
        }

        if( select_object && selection.width > 0 && selection.height
> 0 )
        {
            cvSetImageROI( image, selection );
            cvXorS( image, cvScalarAll(255), image, 0 );
            cvResetImageROI( image );
        }

        cvShowImage( "CamShiftDemo", image );

    //Save the previous centroid to compute the difference
    prevX = centroidX;
    prevY = centroidY;
    //save center coordinates to global variable
    centroidX = track_box.center.x;
    centroidY = track_box.center.y;

    //compute the difference between previous and current centroid
    differenceX = abs (prevX - centroidX);
    differenceY = abs (prevY - centroidY);

    //if difference is too great, then send error values
    if (differenceX > 20 || differenceY > 20)
    {
      cc = -5;
    }
    else
    {
      cc = 0;
    }

    cvWaitKey (10);
```

Appendix A (Continued)

```
     if (c == 27)
     {
       break;
     }
    }

    cvReleaseCapture( &capture );
    cvDestroyWindow("CamShiftDemo");

    return 0;
}
```

Appendix A (Continued)

SIFT Tracking for Grasping Phase Based on (22)

```
/*
Detects SIFT features in two images and finds matches between them.

Copyright (C) 2006-2010  Rob Hess <hess@eecs.oregonstate.edu>

@version 1.1.2-20100521
*/

#include "match.h"
#include "sift.h"
#include "imgfeatures.h"
#include "kdtree.h"
#include "utils.h"
#include "xform.h"

#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

#include <stdio.h>
#include <math.h>


/* the maximum number of keypoint NN candidates to check during BBF
search */
#define KDTREE_BBF_MAX_NN_CHKS 200

/* threshold on squared ratio of distances between NN and 2nd NN */
#define NN_SQ_DIST_RATIO_THR 0.49

/****************************** Globals
**********************************/

//char img1_file[] = "glass.pgm";
//char img2_file[] = "scene1.pgm";
//extern double xdiff, ydiff, ratio;

/******************************** Main
***********************************/

double ** siftMatch(IplImage* img1, IplImage* img2, struct feature**
ffeat1, int *pn1)
{
  struct feature* feat1 = *ffeat1;
  int n1 = *pn1;
  IplImage* stacked = stack_imgs(img1, img2);
  struct feature * feat2, * feat;
  struct feature** nbrs;
  struct kd_node* kd_root;
  CvPoint pt1, pt2;
  double d0, d1;
  int n2, k, i, j, l, adjnf=0, m = 0, mm=0;
```

```
int nminx=1280, nnminx=1280, nminy=1280, nnminy=1280, nmaxx=0,
  nmaxy=0, cenx, ceny;
int flag=0;
double xtot=0, ytot=0, xtota=0, ytota=0;
double *px; //x-coordinates of the goal image
double *py; //y-coordinates of the goal image
double *nx; //x-coordinates of the scene image
double *ny; //y-coordinates of the scene image
double *apx; //adjusted x-coordinates of the goal image
double *apy; //adjusted y-coordinates of the goal image
double *anx; //adjusted x-coordinates of the scene image
double *any; //adjusted y-coordinates of the scene image
double *xd; //differences in x-direction (for e)
double *yd; //differences in y-direction (for e)
double *xx; //differences in x-direction (for x in Lx)
double *yy; //differences in y-direction (for y in Lx)
int *nf; //number of matched features
double **viserv; //pointer to the pointers for visual servoing data
double mincx=1280;
double mincy=1280;
int jxy;

if (!n1)
{
  n1 = sift_features( img1, &feat1 );
}
n2 = sift_features( img2, &feat2 );
kd_root = kdtree_build( feat2, n2 );

px = (double *) malloc (n1 * sizeof (double));
py = (double *) malloc (n1 * sizeof (double));
nx = (double *) malloc (n1 * sizeof (double));
ny = (double *) malloc (n1 * sizeof (double));
nf = (int *) malloc (sizeof (int));
viserv = (double **)malloc((8 * n1 * sizeof (double))+sizeof (int));

for( i = 0; i < n1; i++ )
{
  feat = feat1 + i;
  k = kdtree_bbf_knn(kd_root,feat,2,&nbrs,KDTREE_BBF_MAX_NN_CHKS);
  if( k == 2 )
  {
    d0 = descr_dist_sq( feat, nbrs[0] );
    d1 = descr_dist_sq( feat, nbrs[1] );
    if( d0 < d1 * NN_SQ_DIST_RATIO_THR )
    {
      pt1 = cvPoint( cvRound( feat->x ), cvRound( feat->y ) );
      pt2 = cvPoint( cvRound( nbrs[0]->x ), cvRound( nbrs[0]->y ) );

      //Find min and max values of matched features in the scene
        image
      if (nbrs[0]->x < nminx)
      {
        nminx = cvRound(nbrs[0]->x);
```

```
            }
            if (nbrs[0]->x > nmaxx)
            {
              nmaxx = cvRound(nbrs[0]->x);
            }
            if (nbrs[0]->y < nminy)
            {
              nminy = cvRound(nbrs[0]->y);
            }
            if (nbrs[0]->y > nmaxy)
            {
              nmaxy = cvRound(nbrs[0]->y);
            }
            pt2.y += img1->height;
            cvLine( stacked, pt1, pt2, CV_RGB(255,0,255), 1, 8, 0 );

            //save x- and y-coordinates for goal image
            px[m] = feat->x;
            py[m] = feat->y;

            //save x- and y-coordinates for scene image
            nx[m] = nbrs[0]->x;
            ny[m] = nbrs[0]->y;

            //compute x- and y-differences and update running total for
              average
            xtot = xtot + (feat->x - nbrs[0]->x);
            ytot = ytot + (feat->y - nbrs[0]->y);

            cvCircle(img2, cvPoint(cvRound(nbrs[0]->x),cvRound(nbrs[0]-
              >y)), 1, CV_RGB(255,0,0), 2, 8, 0);
            m++;
            feat1[i].fwd_match = nbrs[0];
          }
        }
        free( nbrs );
      }

      if (m > 0) //if there are some features, then process them and remove
        outliers
      {
        //malloc for adjusted x,y data based on number of matched features
        xd = (double *) malloc (m * sizeof (double));
        yd = (double *) malloc (m * sizeof (double));
        xx = (double *) malloc (m * sizeof (double));
        yy = (double *) malloc (m * sizeof (double));
        apx = (double *) malloc (m * sizeof (double));
        apy = (double *) malloc (m * sizeof (double));
        anx = (double *) malloc (m * sizeof (double));
        any = (double *) malloc (m * sizeof (double));

        //compute the centroid of the scene image features
        cenx = ((nmaxx - nminx) / 2) + nminx;
        ceny = ((nmaxy - nminy) / 2) + nminy;
```

Appendix A (Continued)

```
cvCircle(img2, cvPoint(cenx,ceny), 1, CV_RGB(0,255,0), 2, 8, 0);

//loop through to find feature point closest to centroid
for (j=0; j<m; j++)
{
  if ((abs(nx[j]-cenx) < mincx) && (abs(ny[j]-ceny) < mincy))
  {
    mincx = abs(nx[j]-cenx);
    mincy = abs(ny[j]-ceny);
    jxy = j;
  }
}
//save closest feature point to adjusted points
apx[adjnf] = px[jxy];
apy[adjnf] = py[jxy];
anx[adjnf] = nx[jxy];
any[adjnf] = ny[jxy];
xd[adjnf] = px[jxy]-nx[jxy];
yd[adjnf] = py[jxy]-ny[jxy];
xx[adjnf] = nx[jxy]-160;
yy[adjnf] = ny[jxy]-120;
adjnf++;

//loop through to get rid of outliers
for (j=0; j<m; j++)
{
  for (l=0; l<adjnf; l++)
  {
    if ((nx[j] < anx[l]+20 && nx[j] > anx[l]-20) && (ny[j] <
      any[l]+20 && ny[j] > any[l]-20))
    {
      cvCircle (img2, cvPoint (cvRound (nx[j]), cvRound (ny[j])),
        1, CV_RGB(0,0,255), 2, 8, 0);
      apx[adjnf] = px[j];
      apy[adjnf] = py[j];
      anx[adjnf] = nx[j];
      any[adjnf] = ny[j];
      xd[adjnf] = px[j]-nx[j];
      yd[adjnf] = py[j]-ny[j];
      xx[adjnf] = nx[j]-160;
      yy[adjnf] = ny[j]-120;
      adjnf++;
      break;
    }
  }
}
//adjnf is now the adjusted number of features, apx/y and anx/y
  contain adjusted matched features

nf[0] = adjnf; //save number of matched features

//save pointers into viserv to return to application
viserv[0] = apx;
viserv[1] = apy;
```

Appendix A (Continued)

```
    viserv[2] = anx;
    viserv[3] = any;
    viserv[4] = xd;
    viserv[5] = yd;
    viserv[6] = xx;
    viserv[7] = yy;
    viserv[8] = nf;
  }
  else //give dummy pointer data
  {
    //malloc for adjusted x,y data based on number of matched features
    xd = (double *) malloc (sizeof (double));
    yd = (double *) malloc (sizeof (double));
    xx = (double *) malloc (sizeof (double));
    yy = (double *) malloc (sizeof (double));
    apx = (double *) malloc (sizeof (double));
    apy = (double *) malloc (sizeof (double));
    anx = (double *) malloc (sizeof (double));
    any = (double *) malloc (sizeof (double));

    //give it dummy data
    xd[0] = 0;
    yd[0] = 0;
    xx[0] = 0;
    yy[0] = 0;
    apx[0] = 0;
    apy[0] = 0;
    anx[0] = 0;
    any[0] = 0;

    adjnf = 1;

    nf[0] = adjnf; //save number of matched features

    //save pointers into viserv to return to application
    viserv[0] = apx;
    viserv[1] = apy;
    viserv[2] = anx;
    viserv[3] = any;
    viserv[4] = xd;
    viserv[5] = yd;
    viserv[6] = xx;
    viserv[7] = yy;
    viserv[8] = nf;
  }

  cvNamedWindow("Scene", 1);
  cvShowImage("Scene", img2);
  cvNamedWindow( "Matches", 1 );
  cvShowImage( "Matches", stacked );
  flag = cvWaitKey( 1 );

  cvWaitKey(1);
```

Appendix A (Continued)

```
    cvReleaseImage( &stacked );
    kdtree_release( kd_root );
    free( feat2 );
    *pn1 = n1;
    *ffeat1 = feat1;
    return viserv;
}
```

About the Author

William Pence graduated from Land O' Lakes high school in 2005 and graduated from the University of South Florida in the Fall of 2011 with his Bachelors and Masters in computer engineering. As a research assistant beginning in 2009, he worked with the Center for Assistive, Rehabilitation and Robotics Technologies (CARRT) at the University of South Florida. There, he worked with the WMRA project on many programming and electrical aspects of the project. He currently lives in Tampa, FL and his interests include hunting, fishing, sailing, restoring vintage Jeeps, and robotics. His desire is to work in industry in the fields of robotics and computer engineering.