Real-time Tracking Meets Online Grasp Planning

Danica Kragić,
Computer Vision and Active Perception
Numerical Analysis and Computer Science
Royal Institute of Technology
SE–100 44 Stockholm, Sweden †

Andrew T. Miller and Peter K. Allen
Department of Computer Science
Columbia University
New York, NY 10027 *

Abstract

We present a robotic grasping system that integrates real–time vision with online grasp planning. Grasp planners need to know the pose of objects in a workspace to plan a stable grasp, and real-time visual control systems need to determine how a robot hand should approach, grasp, and move an object to a new configuration. This paper describes a synergistic integration of a grasping simulator and a real-time visual tracking system, that work in concert to 1) find an object's pose, 2) plan grasps and movement trajectories, and 3) visually monitor task execution. Starting with a CAD model of an object to be grasped, the system can find the object's pose through vision which can then synchronize the state of the robot workcell with an online, model-based grasp planning and visualization system we have developed called GraspIt!. GraspIt! can then plan a stable grasp for the object in its current configuration, and direct the robotic hand system to effect the grasp. It can also generate trajectories for the movement of the grapsed object. The grasp and trajectories are then used by the visual control system to monitor the task and compare the actual grasp and trajectory with the planned ones. We present experimental results using typical grasping tasks.

1 Introduction

Visual control of robotic tasks is a major goal of current robotics research. The advent of real-time vision systems has led to the creation of systems that use cameras to control robotic tasks such as grasping [14]. This paper describes a system that extends this paradigm in a number of ways. Most importantly, we have been able to integrate real-time vision with online, model-based simulation, to create a grasp planning, control and monitoring system that can visually determine an object's pose, plan a stable grasp, execute the grasp, and monitor the task for errors and correctness (see Fig. 1). The vision system is responsible for providing accurate and fast estimates of an object's pose, while the grasping simulator uses these estimates to plan 1) an effective and stable grasp of the object and 2) a collision free trajectory in the workspace for transporting the grasped object to a new position. The simulator output is then used to execute the planned grasp and trajectory, which is monitored by the vision system. The vision system can compare the planned trajectory poses with actual object pose as the object is tracked, making sure no errors have occurred, and that the grasp is correct.

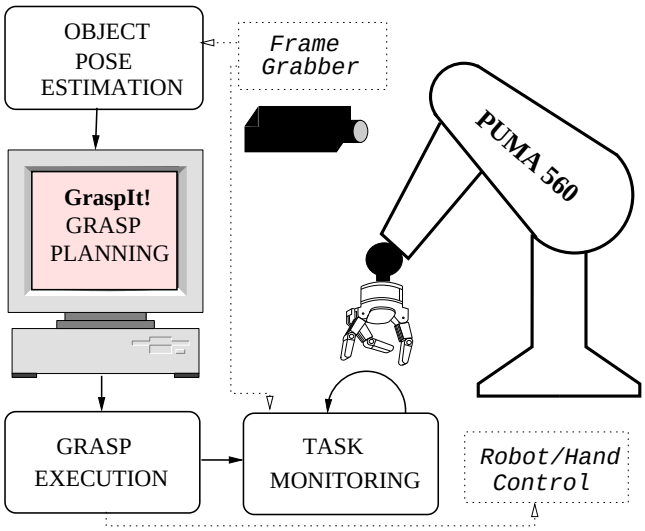


Figure 1: Block diagram of the system.

By merging visual control with grasp planning, we have created a system that can extend the idea of autonomous robot control. Using a calibrated vision sys-

*This work was supported in part by an ONR/DARPA MURI award ONR N00014-95-1-0601 and NSF grant CDA-96-25374.

†This research has been sponsored by the Swedish Foundation for Strategic Research through the Centre for Autonomous Systems. The funding is gratefully acknowledged.

tem, we only need a CAD model of the object to be grasped to both track the object and determine its pose in real-time. The grasping simulator, previously developed by Miller and Allen [10, 11], is able to compute a stable grasp with a quality measure in real-time using the same CAD model. The grasp stability analysis also includes material properties of the object, and the simulator contains a library of robots and hands to choose from, giving it added flexibility for use in many different robot workcells. Below, we describe the vision and grasp simulation modules, and then discuss the integration of these modules on real grasping tasks using a Barrett hand.

2 Pose Estimation and Tracking

Closed-loop control of a robot where vision is used in the feedback loop is commonly referred to as *visual servoing*, [8]. Vision based tracking techniques incorporated in visual servoing systems usually facilitate model based or feature based approaches. A model based approach relies on a CAD model of the object to be tracked [4, 13] while feature based techniques use distinct image features: corners, lines, regions [7]. There are two categories of visual servoing: *position based servoing* and *image based servoing*, [8]. A position based approach usually requires the estimation of a target's pose in the camera or the robot coordinate system. Image based visual servoing uses visual information directly from the image. In our work, we use both approaches: a position based approach is used for driving the arm and the hand to a pose generated by a simulator. On the other hand, image based servo control is used to visually servo a grasped object to the desired pose.

By definition, a necessary requirement of a visual servo system is continuous tracking. Our tracking system is based on the ideas proposed in [4]. The method relies on the assumption of the 2D affine motion model of image features between two successive images. Since this model does not perfectly capture the complex 3D motion, the affine step is followed by an optimization technique where the pose space of the target is searched for the most suitable set of pose parameters given the image data. The general idea and the tracking system operation are presented in Fig. 2.

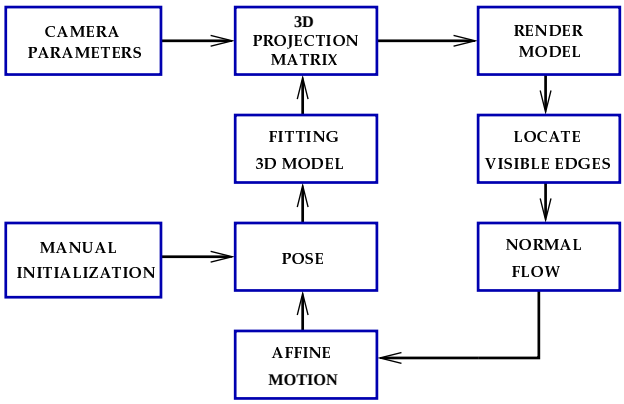


Figure 2: The tracking system.

Pose Estimation

We want to estimate the position and orientation of the object, $\Omega(\mathbf{R}, \mathbf{t})$, with respect to the camera coordinate system. The method we implemented, proposed by Dementhon [2], uses N point correspondences and relies on linear algebra techniques. The method does not require an initial pose estimate or matrix inversions like iterative methods based on Newton-Raphson minimization. It starts by assuming a scaled orthographic projection and iteratively converges to a perspective projection by minimizing the error between the original image points and projected points. This step is followed by an extension of Lowe's [9] nonlinear approach proposed in [6]. The pose estimation step is called POSE in Fig. 2.

The initialization of the tracking system is at this stage done manually. A number of corresponding points on the wire frame model and the image are chosen by the user, see Fig. 3(a) and Fig. 3(b).

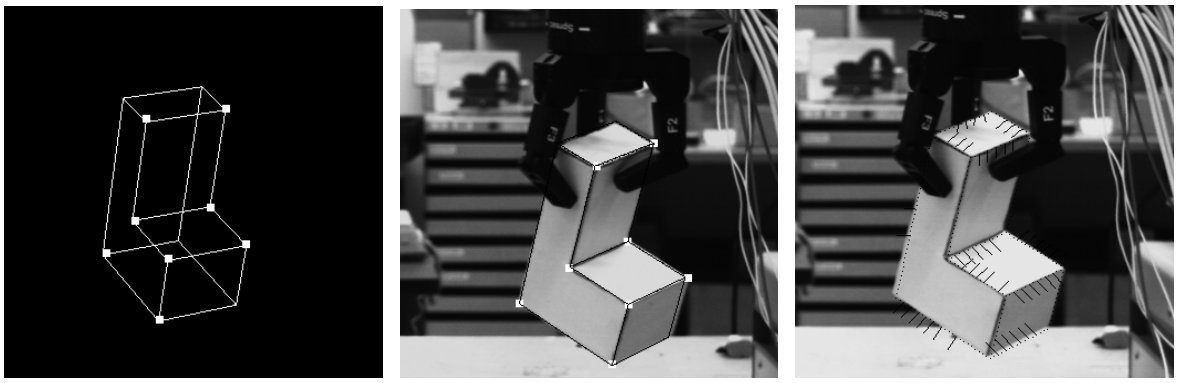
Normal Displacement and Affine Motion

After the pose estimation is obtained, the internal camera parameters are used to project the model of the object on the image plane. A simple rendering technique is used to determine the visible edges of the object. For each visible edge, tracking nodes are assigned at regular intervals in image coordinates along the edge direction. After that, a search is performed for the maximum discontinuity in the intensity gradient along the normal direction to the edge. The edge normal is approximated with four directions: $0, 45, 90$, and 135 degrees. This way we obtain a displacement vector, $d_i^{\perp}(x_i, y_i) = [\Delta x_i \Delta y_i]^T$, representing the normal displacement field of visible edges.

A 2D affine transformation is expressed:

$$\begin{bmatrix} x_i^{t+1} \\ y_i^{t+1} \end{bmatrix} = \begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \end{bmatrix} \Theta = \mathbf{A}(x_i, y_i)\Theta \tag{1}$$

where $\Theta = (a_1, a_2, a_3, a_4, T_x, T_y)^T$ represents the parameters of the affine model. There is a linear rela-



(a) The model with points used in POSE to determine the pose.

(b) Pose estimation (the model is overlayed in black).

(c) Normal flow estimation (the lines represent the direction and not the magnitude).

Figure 3: The key components of the visual tracking system.

tionship between two consecutive images with respect to Θ:

$$d_i(x_i, y_i) = \mathbf{A}(x_i, y_i)\Theta' \quad (2)$$

$$= \mathbf{A}((x_i, y_i))[\Theta - (1, 0, 0, 1, 0, 0)^T] \quad (3)$$

From the previous equation follows:

$$d_i^{\perp} = \mathbf{n}_i^T \mathbf{d}(P_i) = \mathbf{n}_i^T \mathbf{A}(\mathbf{P}_i)\Theta' \quad (4)$$

where $\mathbf{n}_i$ is a unit vector orthogonal to the edge at a point $\mathbf{P}_i$. From Eq. 4 we can estimate the parameters of the affine model, $\widehat{\Theta}'$ using a M-estimator ρ:

$$\widehat{\Theta}' = \arg\min_{\Theta'} \sum_i \rho(d_i^{\perp} - \mathbf{n}_i^T \mathbf{A}(\mathbf{P}_i)\Theta') \quad (5)$$

Using the estimate of points in the image at time $t+1$, the POSE step is performed in order to obtain the pose of the object in the camera coordinate system, $\Omega(\mathbf{R}, \mathbf{t})_{init}$.

3D Tracking

Using the estimated affine parameters $\widehat{\Theta}'$ and positions of edge nodes at time t, we are able to compute their positions at time $t + 1$ from Eq. 1. As already mentioned, the affine model does not completely account for the 3D motion and perspective effects that occur during the object motion. Therefore, the pose space, $\Omega(\mathbf{R}, \mathbf{t})$, should be searched for a best fit given the image data. As proposed in [4], the projection of the object model is fitted to the spatial intensity gradients in the image, using $\Omega(\mathbf{R}, \mathbf{t})_{init}$ as the initialization:

$$\widehat{\Omega}(\mathbf{R}, \mathbf{t}) = \arg\min_{\Omega} \left[-\sum_{C_\Omega} \| \nabla \mathbf{G}(t+1) \| \right] \quad (6)$$

where $\nabla \mathbf{G}(t + 1)$ is the intensity gradient along the projected model edges and C_Ω are the visible edges of the model for given a pose Ω.

The optimization method uses a discrete hierarchical search with respect to the pose parameters. However, the correct discrete step of the pose parameters is crucial in order to find the right minimum value. The affine motion model is particularly sensitive to the rotational motions of the object and large changes in rotation usually result in a loss of tracking. Extensive experiments have shown that the proposed optimization approach is the major bottleneck of the algorithm. For that purpose, we implemented an adaptive search step determination based on the object's 3D velocity. The basic idea is to dynamically change the size of the search step instead of keeping it constant.

We have also implemented a method proposed in [3] that uses the properties of the Lie agebra representation. This technique is more robust in the case where the rotational velocity of the object is significant. Our future work will consider a more thorough analysis of both techniques.

3 Grasping Simulator

GraspIt! is a real-time, interactive simulator that allows the user to manipulate a model of an articulated

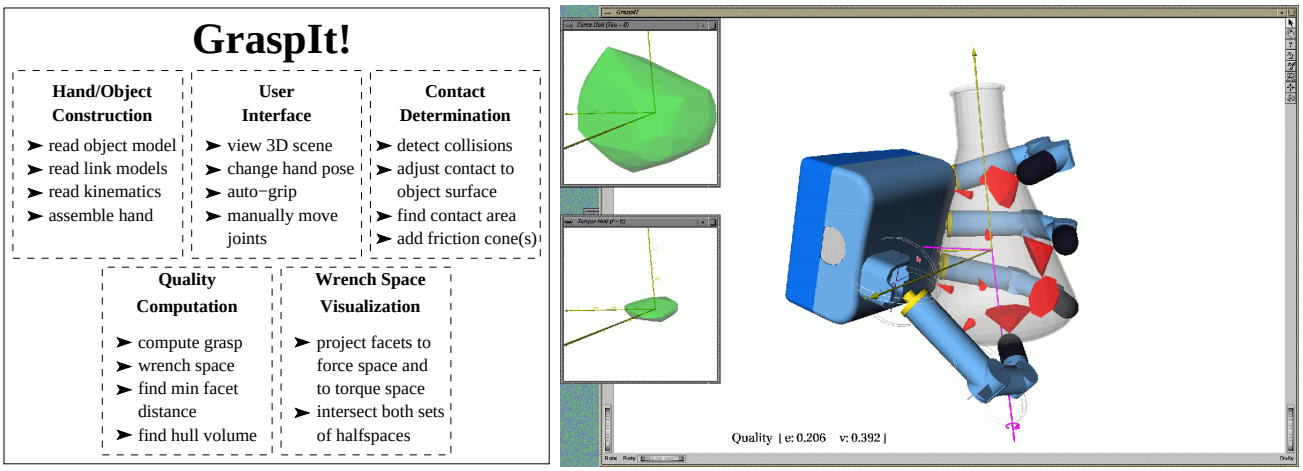


Figure 4: (Left) The internal components of GraspIt! and their functions. (Right) The DLR hand grasping a flask. The volumes in the small windows are projections of the grasp wrench space for this grasp.

robotic hand, perform grasps with the hand, and visualize stability analysis results on the fly. GraspIt! has facilities for modeling a robotic arm and workcell to allow the user to examine the reachability constraints for a given grasping task as well. By consistently using accurate geometric models of all elements of the task, we ensure that grasps planned in the simulator can actually be executed in the real world. In this section we describe how a user interacts with the system and briefly describe some of the components that make up GraspIt! (summarized in Fig. 4). Refer to reviews by Bicchi and Kumar [1] and Shimoga [12] for further information regarding grasping analysis.

First the user must decide which robotic hand and which object should be used in this grasping task. Our library of hands currently includes the Barrett Hand, the DLR hand (shown above), the NASA Robonaut hand, the Rutgers hand, and a simple parallel jaw gripper. While our experiments in this paper focus on the Barrett hand, since it is available to us in the lab, they could be replicated with any of the other hands. Furthermore, by using a simple configuration file that specifies a hand's kinematics and geometry, we have kept the system flexible enough to import new hand designs with ease. An object model can be made from a CAD file, with additional information specifying the surface material and center of gravity. The visual tracking system uses a similar wire frame model representation, which can be derived from the CAD model (see Fig. 3(a)).

Once the hand and object have been chosen, the simulator constructs the models and allows the user to manipulate the hand model. If a connection is established with the object tracking module, the simulator sets the initial pose of the object in relation to the robot. At this point the user begins to plan the grasp by translating and rotating the wrist within the constraints of the Puma arm. Real-time collision detection prevents the hand model from penetrating any other world elements. After positioning the wrist, the user may manipulate the finger degrees of freedom individually, or perform an auto-grasp where the joints are closed at fixed velocities until contact occurs, providing a faster way to perform an initial grasp.

Each time a contact occurs between a link model and an object model, the system determines the contact area and places friction cones at the points of contact. The size of these cones is determined by the two contacting surface materials and a static Coulomb friction model. For tacky surfaces such as rubber, the cone will be wider than for slippery surfaces, indicating that larger tangential forces are possible before slippage occurs, and the grasp will be considered more stable.

The system will analyze the grasp's stability on the fly, as each new contact occurs. Using a convex hull operation, the simulator determines the 6-dimensional space of forces and torques that can be applied by the grasp. This is also known as the grasp wrench space (GWS), and if the space includes the origin, then the grasp is considered stable because it can resist any outside disturbance force by scaling the forces at the contact points. Several metrics have been proposed to quantify the grasp quality, and we currently compute two of them, although others could easily be added. The first is a worst case measure of the grasp's weakest

point, and is defined by the distance from the origin to the closest facet on boundary of the grasp wrench space. Indicators within the scene also show the direction of the worst case disturbance force and torque, and aid the user in finding this weakest point. The second measure is defined by the volume of the GWS and serves as more of an average case quality measure.

In addition to the numeric measures, the user is also presented with various 3-dimensional projections of the GWS to allow better visualization of a grasp's strong and weak points. This information allows the user to incrementally improve the stability of the grasp, and once satisfied, the user may chose to execute the grasp with an actual robotic hand.

4 Experimental Evaluation

To test the system we set up some simple grasping tasks using two different objects. The task involved determing the object's pose from vision, finding an appropriate and stable grasp for the robotic hand given the object's configuration, executing the grasp, and monitoring the grasped object's trajectory as it moved.

4.1 Experimental Setup

The dextrous robot hand used in this example is the Barret Hand. It is an eight–axis, three-fingered mechanical hand with each finger having two joints. One finger is stationary and the other two can spread synchronously up to 180 degrees about the palm (finger 3 is stationary and fingers 1 and 2 rotate about the palm). The hand is controlled by four motors and has 4 degrees of freedom: 1 for the spread angle of the fingers, and 3 for the angles of the proximal links. Our current model of the hand uses slightly simplified link geometries, but the kinematics of the model match the real hand exactly. The hand is attached to a Puma560 arm which operates in a simple workcell. The vision system uses a standard CCD camera (Sony XC-77) with a focal length of 25mm and is calibrated with respect to the robot workspace. The camera is mounted on a tripod and views the robot and workcell from a 2m distance. Both the robot controller and the framegrabber are connected to a Sparc20 workstation which is linked via a TCP socket connection to an SGI Indigo 2 running the GraspIt! software.

The process of planning and executing a grasping task is as follows (refer also to the block diagram presented in Fig. 1):

1. After an image of the scene is obtained, interactive pose estimation of the object is performed as presented in Section 2.
2. A pose estimate is sent to the **GraspIt!** grasp planning software which assists a user to generate the most suitable grasp.
3. Through **GraspIt!** the actual arm and hand are controlled in order to perform the final grasp.
4. Once the object is grasped, the monitoring task begins. Image based visual servoing is used to place the object into the final desired pose. Through **GraspIt!**, a few trajectory points (object poses) are manually generated. Each generated pose is used as an intermediate desired pose for the visual servoing routine.

4.2 Experimental Results

4.2.1 Pose estimation and grasp generation

Our first three example grasps (Fig. 5) demonstrate steps 1-3 of the system (pose estimation, grasp planning and grasp execution). After the first image of the scene is acquired, a manual initialization step is performed as explained in Section 2, and the estimated pose of the object is sent to GraspIt!, which aids the user in selecting an appropriate grasp. After a satisfactory grasp is planned, it is executed by controlling both the robot arm and the robot hand.

Each of the first two examples, show a successfully planned and executed stable grasp of an L-shaped object. The grasp planned in the third example is not stable but requires precise estimation of the object pose so that the thumb can hook through the handle of the mug and it serves to demonstrate the accuracy of the system.

4.2.2 Visual servoing for grasp monitoring

As already mentioned, once the object is grasped the monitoring task begins. We use image based visual servoing to move the object to a desired position. In general, this is setup using a "teach–by–showing" method. In many of the existing systems, [8], the object is first placed into the desired position and feature extraction is performed. After that, the object is put into some initial position and the servoing sequence is initiated by iteratively minimizing the error in the image between the desired and the current image positions of the features. The step of putting the object in the desired position and obtaining the feature measurement is not needed in our case. The user can easily

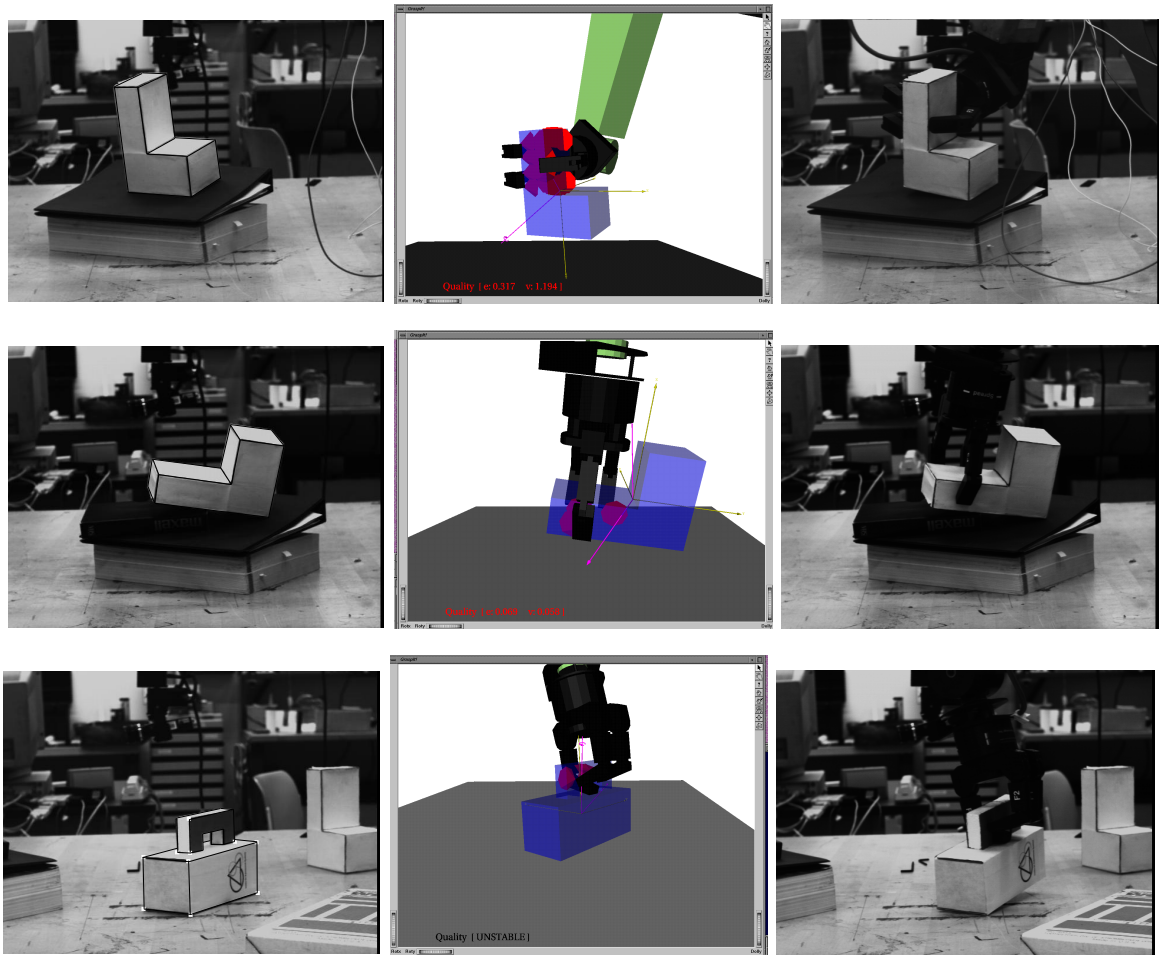


Figure 5: **[Row 1]** Left: The vision system determines the pose of the L-shaped object within the robot workspace. Middle: A stable grasp planned within GraspIt!. Right: The completed grasp. **[Row 2]** The user has successfully planned and executed a stable grasp of the object in a new orientation. **[Row 3]** While the grasp planned in this example is not stable, it demonstrates the accuracy of the system. The camera is 2m away from the workspace, yet the thumb was able to enter the handle opening which is only 1.5 cm wider than it.

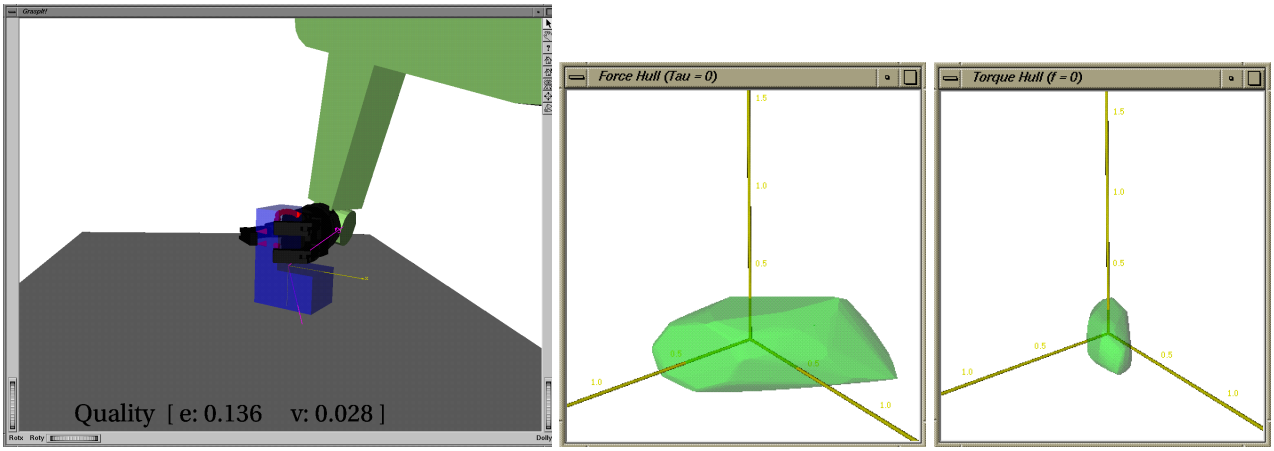


Figure 6: (Left) A stable grasp of the L-shaped object. The axes are positioned at the object's center of gravity (Z axis points into the table). (Middle) The 6D grasp wrench space projected into force space by setting the net torque to 0. (Z axis points up). (Right) The GWS projected into torque space by setting the net force to 0.

specify the object pose at a few different points along the trajectory using the simulator. Using knowledge of the camera parameters and 3D object pose defined within GraspIt!, it is straightforward to estimate the image positions of a number of control points used to construct the error vector.

Briefly, we present the major steps of the image based servo control. As already mentioned, the error $\mathbf{e}$ is to be regulated to $\mathbf{e} = 0$, corresponding to moving the features $\mathbf{f}_c$ to $\mathbf{f}_d$, as defined by:

$$\mathbf{e} = \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = g(\mathbf{f}_d - \mathbf{f}) \tag{7}$$

where g is a positive scalar that controls the convergence rate of a visual servo law. The motion of the image features and the motion of the gripper are related using the image Jacobian in the following way:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \mathbf{A} \begin{bmatrix} \frac{1}{Z} & 0 & -\frac{x}{Z} & -xy & 1+x^2 & -y \\ 0 & \frac{1}{Z} & -\frac{y}{Z} & -1-y^2 & xy & x \end{bmatrix} \begin{bmatrix} \mathbf{R}^{GC} & \mathbf{R}^{GC}\mathrm{S}(\mathbf{t}^{GC}) \\ 0^T & \mathbf{R}^{GC} \end{bmatrix} \begin{bmatrix} V_X \\ V_Y \\ V_Z \\ \omega_X \\ \omega_Y \\ \omega_Z \end{bmatrix} \tag{8}$$

where $\mathbf{A}$ represents the matrix of internal camera parameters followed by a well known image Jacobian matrix for point features. Next is the matrix needed to map the velocities between the gripper and the camera coordinate frame followed by a velocity screw vector of the grippper. This equation is usually written as:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \mathbf{J}\mathbf{T}_G \tag{9}$$

Combining equations (Eq. 7) and (Eq. 9) we obtain:

$$\mathbf{J}\mathbf{T}_G = g(\mathbf{f}_d - \mathbf{f}) \tag{10}$$

The robot velocity screw is computed as:

$$\mathbf{T}_G = g(\hat{\mathbf{J}}^T\mathbf{W}\hat{\mathbf{J}})^{-1}\hat{\mathbf{J}}^T\mathbf{W}(\mathbf{f} - \mathbf{f}_d) = g\hat{\mathbf{J}}^{\dagger}(\mathbf{f}_d - \mathbf{f}) \tag{11}$$

where $\hat{\mathbf{J}}$ is a model of $\mathbf{J}$ which is used in the control expression and $\mathbf{W}$ is a symmetric positive matrix of rank 6 used to select some preferred points out of k available ones.

In this experiment we chose a highly stable grasp around the side of the block, (see Fig. 6). The flat palm of the Barrett hand allows for a strong planar contact on one side of the object and the planar finger surfaces form line contacts when closed around the edges of the opposite face. Looking at the projection of the GWS into force space, shown in the middle of Fig. 6, we see that this grasp can apply strong forces along the horizontal plane but is weaker in the vertical direction since it relies purely on tangential friction forces. Having planned a stable grasp, we executed it from the simulator as before. Then we moved the simulated object to a desired destination pose, as shown in Fig. 7, and sent this pose to the visual servoing system. Using the camera parameters, the corners of the object in its destination pose were projected to the image and can be seen in Fig. 8, which was recorded soon after the start of the task execution. Fig. 9 shows the object having reached its destination.

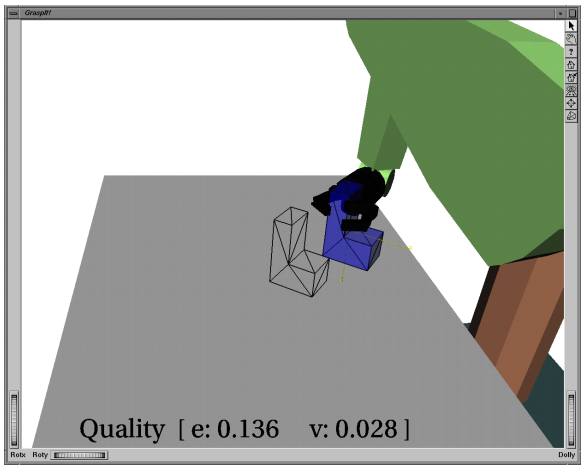


Figure 7: The task is planned in GraspIt! by moving the object from its original location (empty wireframe) to its destination point and saving the new object pose.

5 Summary and Conclusion

We have implemented a robotic grasping system that integrates real-time vision with online grasp planning via a simulator to execute stable grasps on objects, and which can also monitor a grasped object's trajectory for correctness. The system represents a step forward in autonomous control of tasks such as grasping.

Although the system is still in an initial phase, we believe it is an important step in integrating real-time visual control, grasp planning, and online task monitoring. All of the system components are still open research issues in the field of robotics and we will continue to improve the performance of each of them.

The visual tracking component has as its major goals improved robustness and increased speed. The

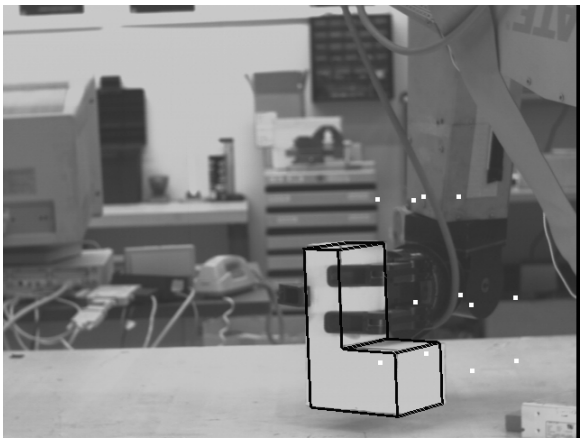


Figure 8: A snapshot of the servoing near the beginning of the task. The object is being tracked and the corners of the destination pose are shown as white dots in the image.

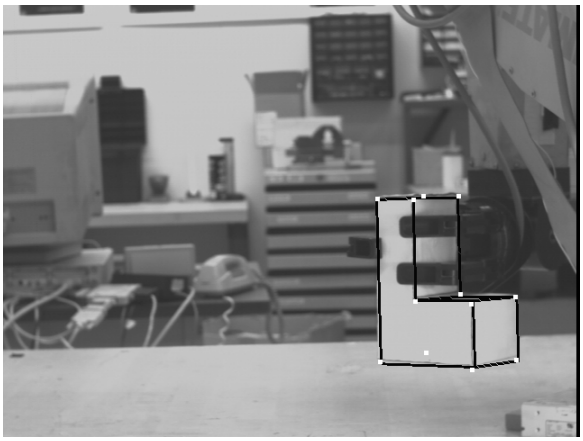


Figure 9: The object has reached its destination pose.

current update rate of 10Hz is far from satisfactory. To improve the robustness in the case of significant oclussions and natural background, we will continue to test the two implemented tracking techniques more extensively. In addition, none of the techniques incorporates any kind of velocity or acceleration prediction which can be easily added (e.g. a Kalman filter approach).

Currently, the grasp planning function is only semi-automated; the user needs to place the hand in proximity of the object and an automated finger contact procedure is instituted. We are exploring adding rules to properly synthesize a grasp given an object's pose and environmental constraints [5].

Acknowledgment: We would like to thank Professor Gerd Hirzinger and Dr. Max Fischer from the German Aerospace Center (DLR) for providing us with models of their robotic hand.

References

[1] A. Bicchi and V. Kumar. Robotic grasping and contact: A review. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 348–353, 2000.

[2] D.Dementhon and L. S. Davis. Model–based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[3] T. W. Drummond and R. Cipolla. Visual tracking and control using Lie algebras. In *Proc of the IEEE Int. Conf. on Computer Vision and Pattern Recognition*, pages 652–657, 1999.

[4] P.Bouthemy E.Marchand and F.Chaumette. A 2D–3D model–based approach to real–time visual tracking. Technical report, INRIA, March, 2000.

[5] K. Goldberg, E. Smith, K. Bohringer, and J. Craig. Computing parallel-jaw grip points. In *Proc of the IEEE Int. Conf. on Robotics and Automation*, 1999.

[6] R. L. Canceroni H. Araujo and C.M. Brown. A fully projective formulation for Lowe's tracking algorithm. Technical report 641, The University of Rochester, CS Department, Rochester, NY, November, 1996.

[7] G. Hager and K. Toyama. The XVision system: A general-purpose substrate for portable real–time vision applications. *Computer Vision and Image Understanding*, 1(69):22–37.

[8] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

[9] D. G. Lowe. Robust model–based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, 8(2):113–122, 1992.

[10] A. Miller and P. Allen. Examples of 3D grasp quality computations. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1240–1246, 1999.

[11] A. Miller and P. Allen. GraspIt!: A versatile simulator for grasping analysis. In *Proc. of the ASME Int. Mechanical Engineering Congress and Exposition*, 2000.

[12] K. B. Shimoga. Robot grasp synthesis algorithms: A survey. *International Journal of Robotics Research*, 15(3):230–266, June 1996.

[13] P. Wunsch and G.Hirzinger. Real–time visual tracking of 3D objects with dynamic handling of occlusion. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 2868–2872, 1997.

[14] B. Yoshimi and P. Allen. Visual control of grasping. In D. Kriegman, G. Hager, and S. Morse, editors, *The Confluence of Vision and Control*, pages 195–209. Springer-Verlag, 1998.