A framework for vision-guided trajectory generation with a 7 DOF robot arm

Jonathan Scholz, Venkata Subramanian, and Arun Kumar Elangovan

Project Report - Robot Intelligence - Planning in Action
Faculty Adviser: Mike Stilman

***Abstract*—This paper describes the development of a software architecture for visually-guided workspace control on a 7 degree-of-freedom manipulator. Beginning with hardware and low-level drivers for the arm, a force sensor, a camera, and a 6-axis mouse, we produced a system capable of tracking and capturing small autonomous mobile robots [3]. The project demonstrates the core capability requirements for *workspace control*, *visual object recognition & tracking*, and *tactile feedback*, all in a modular framework that allows shared state information and structured cooperation between independent components.**

I. Introduction

The primary goal of this project was the development of a control and perception framework for investigating affordance discovery. As currently envisioned, a research program affordance learning will involve tracking sets of visual features as the arm interacts with objects in its workspace. The core features then necessary include a means of controlling the arm in workspace coordinates, processing some form of visual data, and reacting to forces experienced during interaction. This experiment serves to combine visual & tactile data for motion planning, and illustrate the benefits of a cooperative modular architecture. The challenge is to track and capture mobile robotic cockroaches [Figure 1] while avoiding damage to the arm, target, or environment.
Robot "cockroaches", or HexBugs^{TM}, were acquired from RadioShack. These robots are equipped with a pair of metal touch sensors in the position of antennae that allow it to stop and turn when it encounters obstacles. A simple stage for the experiment was constructed that bounded the navigable space of the HexBugs with wooden blocks. This apparatus provided an environment with a predictable set of dynamics which was ideal for the development of the software architecture for the Schunk arm. The goal for the experiment was to visually track the HexBugs after releasing them into the workspace, to apply the proper velocities to the 7 joint motors in the arm to converge upon the bug, and finally capture it with a small plastic cup affixed to the end of the arm.

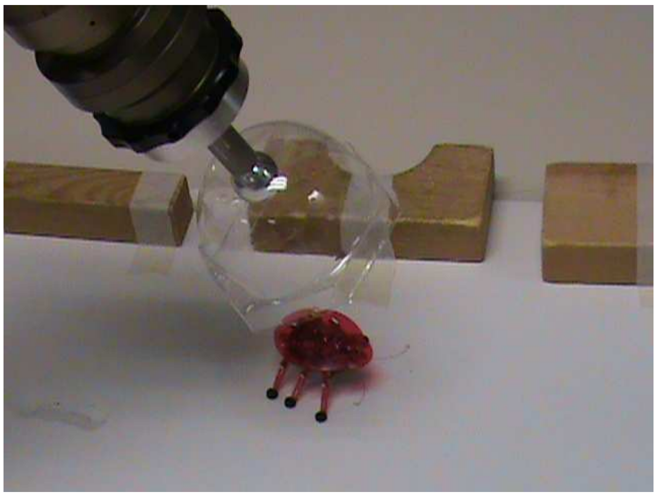


Fig. 1. RadioShack Hexbug^{TM}Model 60-206, seen under Schunk end-effector equipped with plastic trapping cup

II. Related Work

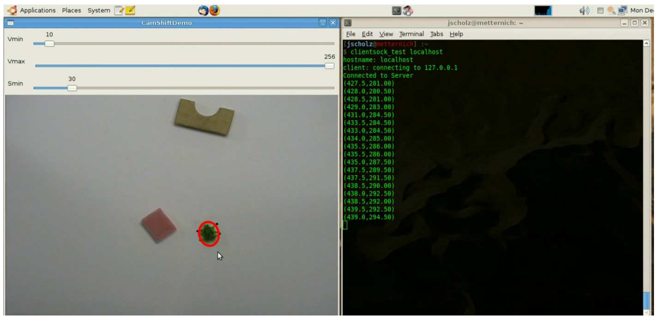
Several other research projects in the area of affordance learning with robot manipulators have used a basic research apparatus like the one described here. Each of these projects share a need for workspace control, tactile feedback, and some form of visual perception. In a project aimed at autonomously discovering object kinematics, Brock et. al. employ a single robot arm equipped with a force sensor, connected to an overhead camera for sensing changes to objects in the workspace [6]. Stoytchev [8] also makes use of an overhead camera and a force-sensitive gripper while investigating tool affordances with a 5 DOF manipulator. It is our goal to merge this work in affordance discovery with recent developments in computer vision on more sophisticated representations of the affordance to visual feature relationship. In his graduate thesis, Sun [9] describes a system for probabilistically relating visual features to categories of traversability affordances. His system naturally depends on relatively sophisticated environment sensing, and we view this project as a step towards that goal.

III. Methods

Tracking and capturing the mobile HexBugs required two types of environment sensing. First and foremost, the system needed a visual object tracking system to report the coordinates of the target as it moved. Image data of the workspace was acquired with a Logitech Orbit webcam [4], and processed with the open-source vision library OpenCV [7] running on Ubuntu Hardy. Tactile data was acquired using a National Instruments^{TM}model NI-6224 [5] data-acquisition device captured with the open-source linux driver Comedi [2].

A. Object tracking

HexBugs were tracked using OpenCV's camShift function, which tracks based upon a hue histogram of an image subregion [1]. This simple approach is nevertheless quite



```
[jscholz@metternich] :~
$ clientsock_test localhost
hostname: localhost
client: connecting to 127.0.0.1
Connected to Server
(427.5,281.00)
(428.0,280.50)
(428.5,281.00)
(429.0,283.00)
(431.0,284.50)
(433.5,284.50)
(433.0,284.50)
(434.0,285.00)
(435.5,286.00)
(435.5,286.00)
(435.0,287.50)
(437.5,289.50)
(437.5,291.50)
(438.5,290.00)
(438.0,292.50)
(438.5,292.00)
(439.5,292.50)
(439.0,294.50)
```

Fig. 2. Object tracking using camShift. Left panel shows outline of tracked HexBug. Right panel shows centerpoint transmitted as ASCII text over TCP/IP socket

robust when used on objects that, like the HexBugs, have a unique color profile compared to other objects in the scene. Tracking the HexBugs with camShift and OpenCV was performed as follows:

Initialization:

1) Grab a frame from the camera stream
2) convert frame to HSV, filtering out the range specified by value and saturation thresholds [1]
3) select a subregion of the image that contains the object of interest (and as little else as possible)
4) set a tracking-box for the object centered on the selected region
5) use selection to create a mask image (used in tracking)
6) calculate a hue histogram of the ROI

Tracking:

1) capture a new frame from the camera stream
2) convert frame to HSV using previous filter values
3) backproject the histogram - use the histogram assign probability values by color to pixels in the image
4) call camShift - starting at previous object location, search backprojection for center of highest probability

Trajectory estimation:

1) calculate a displacement vector describing movement over the past 10 iterations of tracking
2) multiply this vector by a scaling constant, set based upon the reaction speed of the arm
3) add this vector to current position, and draw on current frame
4) transmit head of trajectory vector across TCP socket

Communication & Calibration

After identifying the centerpoint of the HexBugs in pixel coordinates, the next step was to convert these values to workspace coordinates and transmit this information to the arm controller. Since the object tracker and arm controller ran on a separate computers, communication between these machines required a network communication layer running on both the object tracking *server* and the *client* motion planner.

[1]camshift and meanshift work poorly at low values (dark colors), and through trial and error I settled on a conservative value of 80 out of 256

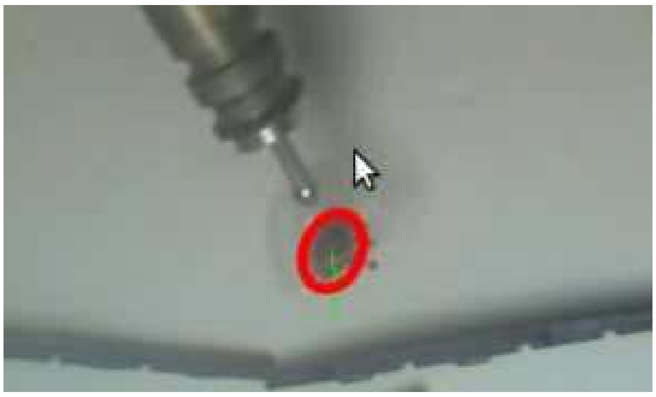


Fig. 3. Trajectory estimate, calculated across past 10 iterations and scaled based on arm reaction time. Green line represents estimate of where to intercept bug

a) Communication: For the purpose of both flexibility and scalability, TCP/IP was used for communication between machines, implemented using a lightweight custom socket library. The object tracking server was configured to transmit centerpoint coordinates, in pixels, as ASCII values to the client machine [Figure 2]. This information was then parsed for values representing the *pixel* location of the target, which was then processed for workspace goal parameters.

b) Calibration v1 - Homography: Given visual input from the camera TCP interface, the system must convert the pixel coordinates representing the target location to the workspace frame of the robot. In the first solution to this problem a simple homography was performed to calculate the relationship between pixel and workspace positions. As camera was mounted at a slightly oblique angle (from overhead), the conversion factor was calculated independently for the two axes. A tele-operation module was used to move the arm about the workspace boundaries to find the coordinates of four pennies placed at the corners. The object tracker was then used to find the pixel values for the four pennies. We then manually found the linear function for each of the axes that produced valid workspace positions from pixel coordinates.

c) Calibration v2 - RPY interpolation: While the homography provided a means of directing the arm about the workspace following motions of the target, it left out an explicit way of altering the other 4 parameters of the workspace goal. This proved to be a significant limitation, as the roll, pitch, and yaw were left fixed to whatever initial values the end-effector had when tracking began. Thus, the algorithm which calculated the inverse kinematics could only solve for the workspace positions featuring those roll, pitch, and yaw parameters, thereby limiting the accessible workspace of the arm. At a minimum, a method was needed to interpolate roll, pitch, and yaw values from X and Y positions. We devised a solution to this problem, however, that had the added benefit of producing a simple procedure for specifying the desired behavior of the arm as it moved about the workspace. Using a similar calibration procedure as above, we set up a matrix of equations for solving all 5 of the parameters of interest (Z solved independently) as a function of both X and Y. The calibration process then involved simply obtaining the pixel and workspace coordinates for the 4 (or n) number of points of interest, arranging the matrices, and solving the system of equations in Matlab to produce the calibration matrix. It was this method that allowed us to specify a more vertical effector position at nearby positions than extended positions simply through training with the teleop controls. This solution also allowed us to move the camera to arbitrary positions and quickly and accurately generate a new calibration matrix.

B. Tactile feedback

Force-feedback at the end-effector of the arm is both an important safety feature and a valuable source of information for a controlling the arm. The force-torque sensor equipped at the end of the arm reports values for displacement and rotation about each of the three principle axes (x,y,z,roll,pitch,yaw). Using libcomedi, the force-torque module was configured to poll the sensor at 1 KHz. These data were made available to the two primary arm controllers to provide a force-based safety cutoff and a method for active compliance when the arm runs into obstacles.

IV. Control System Architecture

At a minimum, controlling the arm in workspace coordinates requires a mechanism to accept six-dimensional workspace coordinates, translate to seven-dimensional joint values, and write the output to a bus connected to the arm. The proposed system for autonomous manipulation and affordance learning is expected to feature several independent components, arranged in a hierarchy, which issue commands to the arm. In addition, after interfacing a six-dimensional mouse to the arm it became apparent that a modular architecture with preemptible commands was highly desirable. For these reasons, the framework put in place for HexBug-tracking segregated the problem into six independent modules [Figure 4]. These modules communicate using POSIX shared memory for IPC. This allows multiple control processes to update a shared resource, where authorization can be managed based on standard Unix permissions.

A. Jointspace control

Low-level communication with the arm was carried out over a CAN bus using modified driver code from Schunk, based on the NTCAN API. The jointspace controller provided a single memory interface for writing reference velocities for the seven modules of the arm. The control loop interfaced with the force-sensor at 1KHz, using reference values and the current arm position to set the joint velocity of each module. If force exceeded a set threshold the velocity gains were set to zero.

B. Workspace control

The majority of the planning and control for the arm concerned the position of the end-effector in workspace coordinates. To accomplish workspace control, we implemented an algorithm to compute the analytical inverse-kinematics of the arm using two wrist points for the six axes, and constrained the 7th (elbow joint) to have the opposite orientation of the end-effector. Reference positions for the end-effector were read from shared memory and translated to desired joint values, after applying an offset derived from the instantaneous readings from the force-sensor.

C. force-torque API

Translation and rotation values were read from the NI-6224 device using an interface to libcomedi. These values were maintained in a shared memory buffer and updated at 1 KHz.

D. object tracker

Due to the system load for capturing frames from the webcam and running camShift, visual tracking of the HexBug was run on a separate machine. After initializing the tracker and connecting all clients, the centerpoint of the HexBug was transmitted over a TCP socket to the machine controlling the arm.

E. tele-operation

The first demonstration of workspace control on the arm used a six-dimensional mouse with axes mapped directly to the six axes of the end effector. This read workspace coordinates from shared memory, polled the axes of the mouse, and updated the workspace reference point with these displacements.

F. HexBug-Catcher

Packets from the object tracker were collected on the controller machine and parsed for position values of the HexBug. These values were then translated from pixel coordinates to workspace coordinates and written to the shared memory location for the workspace reference point.

G. Performance

After releasing a bug onto the workspace, the arm was able to successfully maneuver its end effector to hover over the same x,y coordinate in real-time [Figure 10]. Due to an unresolved conflict between the CAN bus and the NI-6224 drivers, the arm is not currently able to use force-torque feedback to capture the bug. In addition, the object tracker fails when the arm occludes the bug, and a workaround involving trajectory extrapolation has not been implemented yet.

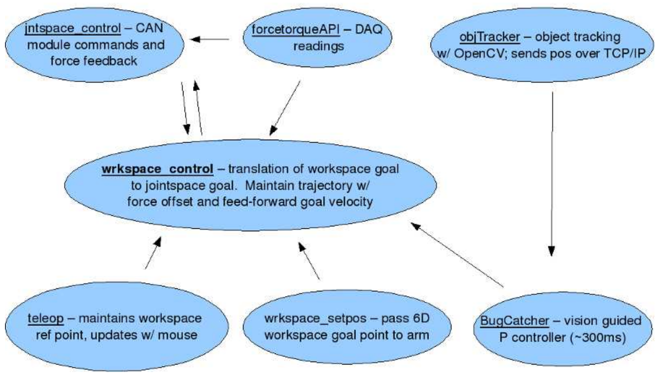
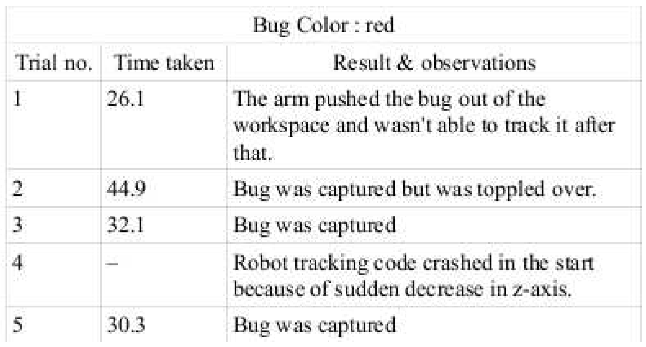

jntspace_control – CAN module commands and force feedback
forcetorqueAPI – DAQ readings
objTracker – object tracking w/ OpenCV; sends pos over TCP/IP
wrkspace_control – translation of workspace goal to jointspace goal. Maintain trajectory w/ force offset and feed-forward goal velocity
teleop – maintains workspace ref point, updates w/ mouse
wrkspace_setpos – pass 6D workspace goal point to arm
BugCatcher – vision guided P controller (~300ms)

Fig. 4. Schematic of the control architecture. Includes six modules for the bug-tracking experiment and an additional mechanism for manually specifying workspace goal coordinates



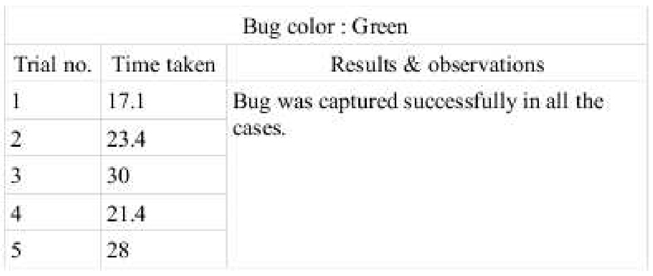
Bug Color : red		
Trial no.	Time taken	Result & observations
1	26.1	The arm pushed the bug out of the workspace and wasn't able to track it after that.
2	44.9	Bug was captured but was toppled over.
3	32.1	Bug was captured
4	–	Robot tracking code crashed in the start because of sudden decrease in z-axis.
5	30.3	Bug was captured

Fig. 5. Experimental observations for ’red’ colored bug

Despite these shortcomings, the arm has consistently been able to maintain an end-effector velocity to match the bug, and we expect a solution to the remaining issues in the near-term.

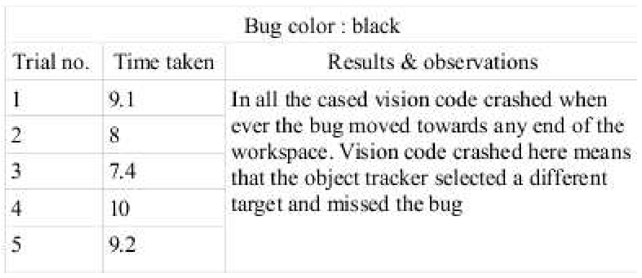
comparison of different targets: Since there is not much planning involved the experiments were devised to test the computational complexity or completeness or optimality of the algorithms. Rather they were devised to check the time taken and efficiency of bug tracking and accuracy of the vision code. There were no special experimental set-up rather the bug was let to move around in the workspace and the time taken by the robot to capture it was recorded. This was done for five trials for a bug and the entire procedure was repeated for four different colored bugs for testing the integration of vision code and the robot controller code. [Figure 5,6,7,8]

1) CAN communication speed: We profiled the communication and command execution part of the arm. We calculated the time it take from starting to send the first CAN message, the arm executing those commands, till the time it takes to recieve the last acknowledgement. We did this profiling in the regular controller, where all the modules are active and commands are sent to all modules. We write the commands to the driver asynchronously and read back the acknowlegement through in blocking mode. We operated the arm at a baudrate of 1000 (1Mbps), the highest the arm can support. We observed the following results. Here each value corresponds to the time it takes to write the commands, the commands to be executed and the acknowledgements to be obtained from the modules. min: 1681.8699999999999 max: 2406.7800000000002. avg: 1844.798365999998. These values were calculated over 10000 data points [Figure 9].



Bug color : Green		
Trial no.	Time taken	Results & observations
1	17.1	Bug was captured successfully in all the cases.
2	23.4	
3	30	
4	21.4	
5	28	

Fig. 6. Experimental observations for ’green’ colored bug

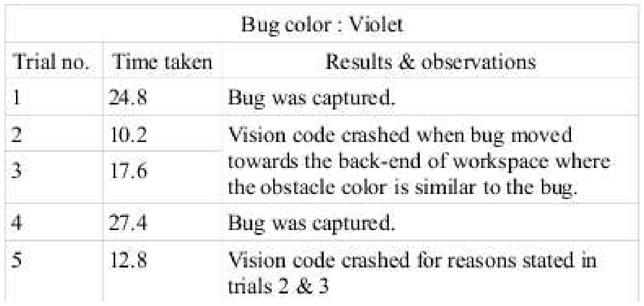


Bug color : black		
Trial no.	Time taken	Results & observations
1	9.1	In all the cased vision code crashed when ever the bug moved towards any end of the workspace. Vision code crashed here means that the object tracker selected a different target and missed the bug
2	8	
3	7.4	
4	10	
5	9.2	

Fig. 7. Experimental observations for ’black’ colored bug

H. Discussion & future directions

Currently, the method we use to calculate the distance between the bug and the arms’s end effector is as follows: we first calculate the joint space configuration of where the end-effector should be (goal position) to catch the bug and its current joint space configuration. We then calculate the euclidian distance between these two to find the remaining distance. But, we still need to figure out a method for calculating the goal configuration and current configuration in workspace. This will help us in cases where we need to track multiple bugs



Bug color : Violet		
Trial no.	Time taken	Results & observations
1	24.8	Bug was captured.
2	10.2	Vision code crashed when bug moved towards the back-end of workspace where the obstacle color is similar to the bug.
3	17.6	
4	27.4	Bug was captured.
5	12.8	Vision code crashed for reasons stated in trials 2 & 3

Fig. 8. Experimental observations for ’violent’ colored bug

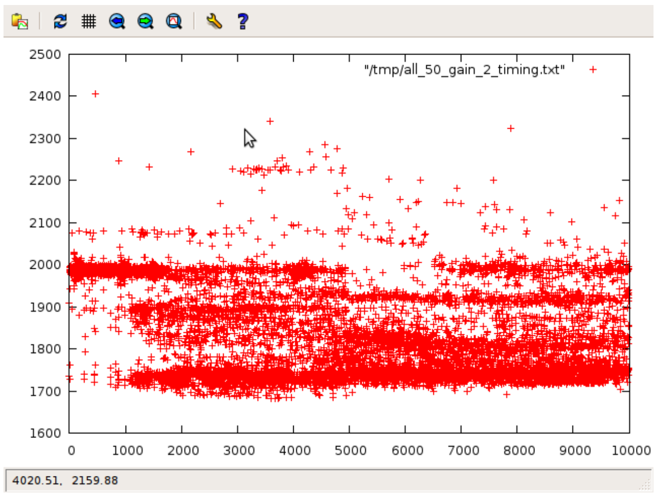


Fig. 9. CAN module communication/execution timing

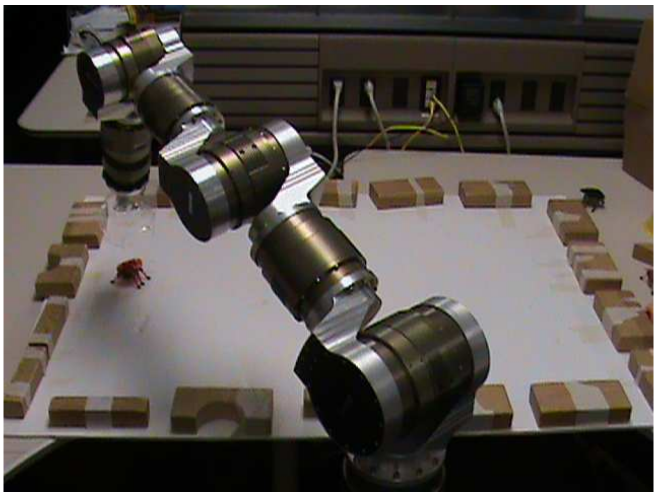


Fig. 10. Still image of robot arm converged on the x,y coordinate of a HexBug while moving in the workspace. Capture with the plastic cup pending resolution of conflict between NTCAN and Comedi

at the same time. Workspace configuration will be a more correct estimate of finding the closest bug in this case, as jointspace configuration distance between the end effector and all bugs is not equivalent to workspace configuration distances.

a) : We also need to work on calculating trajectories in jointspace configuration. This will help us move between current jointspace position and goal jointspace position which are far apart while maintaining a consistent velocity throughout the path and help us move the arm in a predictable fashion. Otherwise, the arm could take a trajectory which might go through the table and thus making it collide with the table. Right now, we avoid it by making the arm's position in the workspace configurations that it will be in to be high enough to make it not hit the table and reduce it only when we are sure of it catching the bug in the process.

b) : The vectors of the forces as sensed by the force sensor at the end effector is currently independent of the orientation of the end effector. We need to transform one to the other so that the arm can back out in the direction opposite to the direction of the force experienced irrespective of its orientation.

c) : Right now, we send velocity commands to the arm to control it. This only indirectly controls the arm as the commands are internally processed by a velocity controller which applies a present damping to the arm. But, sending velocity commands helped us to quickly move on to the other parts of the project. We need to write a controller which controls its motion in terms of torque by controlling the current passing through each of the modules.

References

[1] J.G. Allen, R.Y.D. Xu, and J.S. Jin. Object tracking using CamShift algorithm and multiple quantized feature spaces. In *Proceedings of the Pan-Sydney area workshop on Visual information processing*, pages 3–7. Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 2004.

[2] comedi.org. Comedi - linux control and measurement device interface. http://www.comedi.org, [Accessed: 11/2008].

[3] HexBug Inc. Hexbug micro robotic creature. http://www.hexbug.com, [Accessed: 11/2007].

[4] Logitech Inc. Logitech orbit webcam. http://www.logitech.com/index.cfm/webcamcommunications/orbit, [Accessed: 09/2007].

[5] National Instruments Inc. National instruments ni-6224. http://sine.ni.com/nips/cds/view/p/lang/en/nid/14134, [Accessed: 06/2008].

[6] D. Katz and O. Brock. Extracting Planar Kinematic Models Using Interactive Perception. In *RSS Robot Manipulation workshop*. Springer, 2007.

[7] opencv.org. Opencv - computer vision library. http://sourceforge.net/projects/opencvlibrary/, [Accessed: 11/2008].

[8] A. Stoytchev. Behavior-Grounded Representation of Tool Affordances. In *Robotics and Automation, 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3060–3065, 2005.

[9] J. Sun. Object Categorization for Affordance Prediction.