Playing Chess with a Human-Scale Mobile Manipulator

Michael Ferguson, Kim Gero, Joao Salles
Department of Computer Science
University at Albany
Albany, NY 12222

James Weis
Department of Information Studies
University at Albany
Albany, NY 12222

Abstract

This paper describes our efforts preparing a mobile manipulator for the 2011 AAAI Small Scale Manipulation Challenge. We describe our approach to building a low-cost mobile manipulator for human-scale environments using ROS and off-the-shelf sensory and servos.

Introduction

Our robot (Maxwell) is a low-cost, human-scale mobile manipulator which has been prepared for the 2011 AAAI Small Scale Manipulation Challenge. The robot has a single arm and differential drive mobile base. Sensory consists of a Microsoft Kinect on a pan/tilt stage. Control software was written with the help of the Robot Operating System (ROS).

Hardware

Maxwell carries a laptop on his rear deck, which connects to motors and servos through a USB connection to an ArbotiX2 RoboController. The ArbotiX2 is an AVR-based co-processor which handles all real-time operations.

Mobility is provided by a differential drive base with two independently controlled wheels. A closed loop PID speed control is implemented on the co-processor. The head and arm are raised above the base by a column constructed of 8020 aluminum. The column is constructed of 3 separate pieces, allowing Maxwell to be broken down into pieces with no dimension larger than 20 inches, allowing for easy shipment to the AAAI Challenge.

Primary sensory is from a Microsoft Kinect sensor, mounted on a pan and tilt neck which provides a wider field of view. The Kinect generates both traditional RGB images and depth images which are used to construct a point cloud representation of the environment.

Maxwell's arm consists of a 5 degree of freedom (DOF) manipulator with a 2-servo gripper. The arm is constructed of various-sized Dynamixel servos, which have a robust and easy to use serial bus architecture, and yield a maximum payload of approximately 50 ounces when the arm is fully extended.

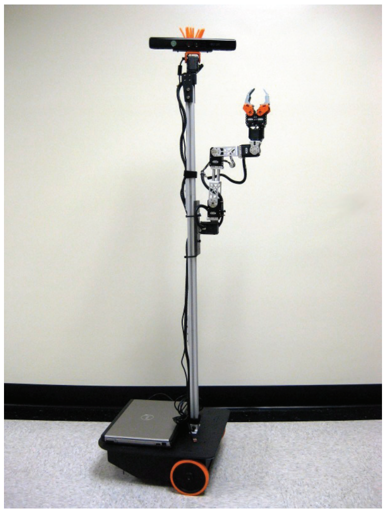

Figure 1: Maxwell, a human-scale mobile manipulator.

Software

Our software was developed with the help of the Robot Operating System (ROS) (Quigley et al. 2009). By using ROS, we gained a head start on many of the challenges of mobile manipulation.

For overall task control we used SMACH, which allows the creation of state-based executives in the Python programming language. Our executive consists of a hierarchical state machine, and is shown in figure 2. The outer level consists of only two states: *my_move* and *your_move*. The *your_move* state simply waits until it is informed that either the opponent has made a move, or the game is over. Making a move is handled by perception and manipulation pipelines within the *my_move* state, which are described below.

Perception

Perception is achieved using a Kinect depth camera, which provides both a traditional RGB image as well as a depth image. The depth image is then converted into an n-dimensional point cloud representation to allow geometric computation.

A background task, the *chess_board_locator*, continuously localizes the robot against the chess board, by com-

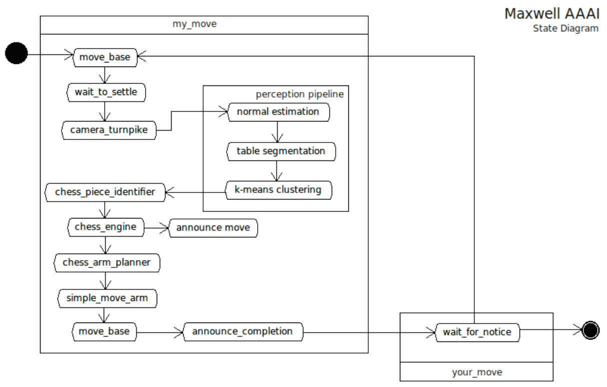

Figure 2: State diagram for chess executive.

puting a transformation between the sensor frame and the lower corner of the chess board. The locator works by finding the lines of the chess board using a Hough transform and then projecting the intersections of these lines into 3d using the point cloud data. We then find the transform by aligning with an ideal board using Iterative Closest Point (ICP).

We then use the Point Cloud Library (PCL) (Rusu and Cousins 2011) to segment the table and chess pieces and find grasp locations. The point cloud processing estimates point normals from which it can approximate a best fit plane and then segment the table points from the chess pieces. After removing points corresponding to the table, segmentation of individual chess pieces is achieved by k-means clustering.

For recognizing different pieces, we are developing a SURF-based recognizer, which works by projecting SURF features into 3D using the point cloud data and the recognizing individual parts. After aligning small point clouds with their SURF recognized model, we can compute a representation of the board which can be passed into a chess engine, GNU Chess. The output of the chess engine is then a move to execute.

Manipulation

The manipulation pipeline converts the abstract plan generated by the chess engine into actual movements of pieces through the 3d coordinate system of the arm. Additional information, such as the type of piece being grasped, can assist in this arm path planning.

We use the existing ROS *arm_kinematics* package to compute inverse kinematics (IK) solutions. However, as our manipulator has only 5 degrees of freedom, we cannot reach all 6-DOF poses within our workspace. The X, Y, Z coordinates of the piece are therefore converted into a 6-DOF pose by several heuristics:

- The yaw angle of the grasper is determined uniquely from X and Y coordinates of the grasper.
- The roll angle of the grasper can be specified from the piece and knowledge of usable grasps.
- The optimal pitch angle of the grasper is an overhead grasp, however we can search the IK configuration space within a small region around overhead grasping.

Once an IK solution for the final grasp has been found, we generate a trajectory from the current gripper location to the grasp point. To reduce the complexity of planning, we do most of our movement in a horizontal plane far enough above the board that we have no collisions with pieces. This makes our search for smooth trajectories perform quicker than a completely unconstrained search. Further, we avoid the overhead of collision-free trajectory planning using more traditional voxel grids.

Calibration

Complex mobile manipulators often face issues with calibration. Tolerances in part manufacturing and assembly quickly add up to disastrous levels of error. It is important then that our sensors and manipulators are *calibrated* to remove the introduced errors between kinematic frames.

Maxwell has additional issues, since he can be disassembled for shipping and transport. The robot must be recalibrated after each travel and reassembly. Our calibration routine uses the point cloud data from our Kinect sensor, which is highly reliable with a small amount of approximately Gaussian noise, and scores particular calibration parameters based on the intersection of Kinect point cloud data and geometric models of the robot and environment. Calibration begins by finding the constraints of the neck pan and tilt joints by a Monte Carlo search until the output of the Kinect matches closely the representation of the floor and robot base. Once the neck has been calibrated, we move to calibrating the arm to the sensor frame using a method similar to (Pradeep, Konolige, and Berger 2010).

Conclusion and Future Works

We have presented here our work on the 2011 AAAI Small Manipulation Challenge. In preparing for this Challenge we have developed a number of reusable modules for our robot, such as manipulation primitives and calibration techniques. In the future we hope to adapt Maxwell to compete in the RoboCup@Home contest by adding an actuated vertical trolley to the arm, allowing the robot reach objects on all surfaces, including the ground, allowing for more general mobile manipulation.

References

Pradeep, V.; Konolige, K.; and Berger, E. 2010. Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *International Symposium on Experimental Robotics (ISER)*.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. ROS: an open-source robot operating system. In *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*.

Rusu, R. B., and Cousins, S. 2011. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation (ICRA)*.