Robotic Arm Control With Human Interactionl

A project of the 2016 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University

https://robotics.shanghaitech.edu.cn/teaching/robotics2016

Huangjie Yu
yuhj@shanghaitech.edu.cn

Xin Chen
chenxin2@shanghaitech.edu.cn

Xuhui Liu
liuxh@shanghaitech.edu.cn

Abstract

We proposed a programmable system that control robotic arm to grasp objects with human interaction. We are able to detect multiple objects of various types. Our system invloves hand-eye calibration which determines the camera pose, vision detection which extract representative positions of objects, preprocessing which sets everything fine for vision algorithm, underlying control which actually move the arm and human interaction which allows the robotic arm to mimic human arm movement. Our program is implemented on ROS, the most popular platform for interacting with robots.

1. Introduction

We proposed a system made up of a robotic arm, a depth camera and leap motion, aiming at grasping multiple objects with human interaction. It's a fundamental task for robotic arm to grasp objects for various applications, e.g. artificial intelligence. Further more, we make use of leap motion for human interaction, allowing robotic arm to mimic human arm.

Our system mainly consists of three parts, vision detection, mechanical control and human interaction. Vision detection was used to detect objects and extract representative point. Mechanical control is responsible for moving the arm correctly and precisely. Human interaction is another functionality that let the robotic arm mimic the movement of a human arm.

2. State of the Art

2.1. Related work

B. Tondu, et al., presented a design of a 7R anthropomorphic robot-arm entirely actuated by antagonistic McKibben artificial muscle pairs [1]. The validation of the tobotarm architecture was performed in a teleoperation mode. Another architecture of a robotic servo system was proposed by K. Kosuge et al. [2]. The controller with proposed architecture can automatically modify the trajectory of a robot arm according to the control strategy represented by a virtual internal model.

Kok-Meng Lee, et al., presented an alternative design of a three-degrees-of-freedom manipulator based on the concept on an in-parallel actuated mechanism [3]. The manipulator has two degrees of orientation freedom and one degree of translator freedom. The basic kinematic equations for use of the manipulator are derived and the in uences of the physical constraints on the range of motion in the practical design are discussed. Several possible applications which include the in-parallel mechanism as part of the manipulation system are suggested. And in this paper, the authors present the kinematic equations for use of a three-degrees-of-freedom in-parallel actuated mechanism as a robot manipulator. The physical constraints imposed by the limits of the ball joints and the link lengths have been discussed. A simulation program has been developed to predict the range of motion for the purpose of practical design. Various possible applications of the in- parallel mechanism as part of the six-degrees-of-freedom manipulator are addressed. Future work should include dynamic analysis, prototype design, and evaluation in an industrial environment and computer-control scheme development.

D. Bassily, el al., discussed the applications of the human-robot interaction [4]. The research of this interaction focus on Ambient Assisted Living AAL which promises to address the needs of elderly people, so the implementation of a intuitive and adaptive manipulate scheme is proposed by developing a human- machine communication interface between the Leap Motion Controller and the 6-DOF Jaco robotic arm. Ashutosh Saxena, et al., presented a learning algorithm for grasping novel objects [5]. It neither requires, nor tries to build, a 3-d model of the object. Train is supervised using synthetic images for the training set.

Sergey Levine, et al., describe a learning-based approach to hand-eye coordination for robotic grasping from monocular images using a CNN [6]. The network observes the spatial relationship between the gripper and objects in the scene, thus learning hand-eye coordination. They use this network to servo the gripper in real time to achieve successful grasps. For the training parts, they collected over 800,000 grasp attempts over the course of two months, using between 6 and 14 robotic manipulators at any given time, with differences in camera placement and hardware. The evaluation demonstrates that this method achieves effective real-time control, can successfully grasp novel objects, and corrects mistakes by continuous servoing.

Moran, et al., described a history of robotic arms [7].The most obvious method in robotic arm evolution was adapted along the lines of human anatomy and kinesiology. So recent robotics arm contains 4 parts: shoulder joint, elbow joint, wrist joint and hand. Karbasi, et al., presented a distance technique for hand detection with Microsoft Kinect [8]. First, Kinect sensor was used to obtain depth image information. Second, background subtraction and interactive method for shadow removal are applied to reduce noise from the depth image. Then it uses Microsoft SDK for extraction process and segmentation.

S. Rosa, et al., implements a system containing a 7-DOF robotic arm for object detection, learning and grasping using vocal information [9]. The program was implemented in ROS and was made up of six nodes: manager node, Julius node, move node, PCL node, festival node and compute node. The information stream starts from Julius node and ends with the move node, managed by manager node. The Julius node is responsible for interpreting user vocal commands. The PCL node implements object recognition. The festival node interacts with the users with voice. The computer node computes the 3D pose of the object for grasping. The move node implements path planning based on the MoveIt package for ROS.

They used Julius for interpreting user commands and vocal feedback. For detection, they first reconstruct point clouds using MeshLab and Poisson Surface Reconstruction; then the scene was segmented and objects are extracted using Euclidean clustering step. A path planning algorithm together with an inverse kinematic solver has been used for the generation of suitable trajectories to move the robotic arm from its starting pose to the target pose.

2.2. Mean Shift

Mean Shift is a very effective algorithm for classification and object tracing [10]. Oliver Kroemer et al. use this algorithm to implement robot grasping in Active Learning using Mean Shift Optimization for Robot Grasping. They suppose a new robot learning framework for reproducing the ability which is learning point of view in robot grasping. To implement this, at first,they let robot observe a few good grass by demonstration and learns a value function for these these grasps using Gaussian process regression. Then, choose grasps which are optimal with respect to this value function using a mean-shift optimization approach and tires them out the real system. Finally, upon every completed trial, the value function is updated, and in the following trials it is more likely to choose even better grasping points. We don't use mean shift for optimization like the algorithm of this paper, but we use mean-shift to classify these point cloud accurately.

Ferran RIGUAL et al. use RANSAC with Mean Shift clustering to improve the performance of their algorithm in the case of multiple instance recognition in [11]. In their work they address the problem of object detection for the purpose of object manipulation in a service robotics scenario. They implement several state-of-the-art object detection methods, and select the best performance. During the evaluation, they find three main practical limitations of current methods which are identified in relation with long-range object detection, grasping point detection and automatic learning of new objects; and propose practical solutions.

2.3. ROS packages

2.3.1 Schunk canopen driver

schunk canopen driver privides a driver interface for the Schunk LWA4P robot arm through the CANOPEN interface [12]. It also provides a simple interface that accepts position commands and another interface for ros control. The package offers two different interfaces, which both differ in the action topics, parameters and the way commanded waypoints are interpreted by the hardware.

The simpler profile-position-interface accepts series of waypoints which the robot will drive to with internal interpolation. It is not guaranteed (and might almost never happen), that all joints finish moving at the same time. If multiple waypoints are given, a new waypoint is started for all joints at the same time as soon as the last joint reached it's previous destination. Ramping up and down between waypoints is done by the robot internally depending on it's configuration. The ramping setup will be treated later on.

The ros-control-interface provides a position controller in joint space which will interpolate between waypoints inside the controller (on the host PC). You can set joint velocities and time constraints for each waypoint which the controller will take care of.

2.3.2 Leap motion

leap-motion is a ROS driver for interfacing with the Leap Motion 3D gesture sensor [13]. The Leap is capable of tracking the fingers and palm of a user with millimeter accuracy. The sensor is capable of producing the following information:3D position for palms of both hands and all 10 fingers, 3D direction vector for all fingers and hands, 3D normal vector for palm.

Through Leap-ROS, you can find the pose of each hand which has be recognized about 90HZ. Then I will focus on the introduction on the way to using this package in detail. First, Install the driver and SDK from manufacturer's site. Then, start the Leap Motion driver.

Thel leap motion driver utilizes the offcial Leap SDK for Linux in order to talk to the device. Currently it does not have all the capabilities the SDK provides, but it is easy to extend. The driver currently provides the following: Single hand 3D Palm Position, Normal Vector, Pose (pitch, yaw and roll), Single hand 3D Hand Direction Vector and All Finger joints [thumb, index, middle, ring, pinky] positions. This information is encompassed in the custom message leap motion/leapros.msg.

2.3.3 Openni2 launch

openni2 launch contains launch files for using openni-compliant devices in ROS [14]. It supports Asus Xtion, which is the device we are using, Xtion Pro, and multiple versions of the Primesense 1.08 and 1.09 cameras.

Openni2 provides a simple access to its rgb and depth information simply by subscribing a certain topic. In our project, we have used point cloud information which is also provided by default.

3. System Description

Our system consists of three separating parts: vision detection, mechanical control and human arm interaction. The vision detection part was used for finding objects and extracting representative points of the objects. The mechanical control part do the underlying moving. And human interaction function uses human arm gesture to control the robotic arm.

I will introduce the pipeline of out system. First, our program will, at some point, yields a point under the world coordinate system. This point is assumed to be close to the object, but not on it. It is recognized by the low-level controlling program who will give moving commands to the arm. Then the arm will move to that point. Now that the palm of the robotic arm is close to the object, we start controlling the arm with our own arm. The movements of our hands are calculated using leap motion, and are given to the controlling program who will again publish control commands to the robotic arm. Thus, it can mimic the gesture of human hands and grasp objects.

Before further processing, we first calibrate the depth camera and the palm of the robotic arm. This gives the transformation from camera to the palm. Since the transformation from palm to base, which determines the world coordinate system, is known in ROS, we are now able to transform points in camera frame to world frame.

Our working space is constrained on a table, the robotic arm is put near one edge of it and objects can be put anywhere on the table. The depth camera was fixed on the arm palm. To start with, we will set the robotic arm to a 'comfortable' position from which it is able to have a complete overview of the whole table and try not to include other objects.

The core vision algorithm in our system is **Mean Shift**, used for localize multiple objects. It is explained in detail in the next section.

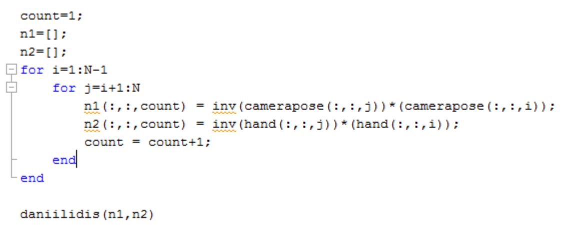
Once we are able know the rough center for each object, we can tell **Moveit!** to plan a path. If no collision is detected, the robotic arm should know how to move there. To perform grasping, however, we need to place the finger exactly at the position of the object. To achieve this, we put a leap motion at the corner of the table. Leap motion can tell the movements of our hands. These movements are then used for calculating goal positions which are sent to the arm. The results look as if the robotic arm is mimicking the human arm.

There are difficulties though. We are unable to control the finger of the robotic arm which makes it impossible to actually grasp objects.

3.1. Program description

3.1.1 Hand-eye calibration

We wrote Matlab codes for calibrating camera and arm palm. The above code snippet gives an overall looking of our algorithm. *camerapose* represents camera pose under the coordinate system determined by check board. *hand*



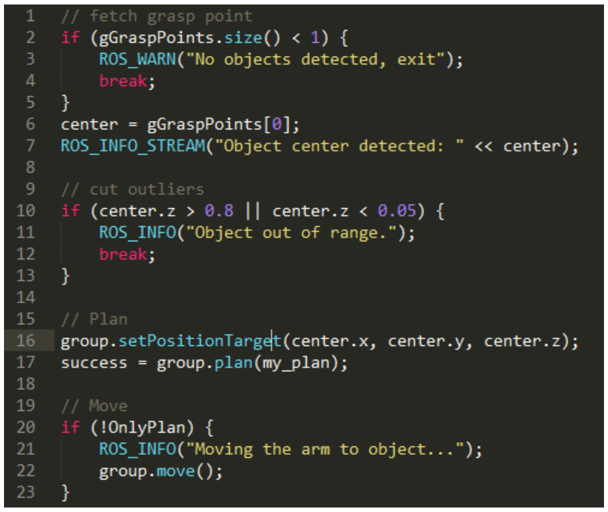
```
count=1;
n1=[];
n2=[];
for i=1:N-1
    for j=i+1:N
        n1(:,:,count) = inv(camerapose(:,:,j))*(camerapose(:,:,i));
        n2(:,:,count) = inv(hand(:,:,j))*(hand(:,:,i));
        count = count+1;
    end
end

daniilidis(n1,n2)
```

Figure 1. Calibration Code

represents hand pose under the coordinate system determined by arm base. The inverse operation gives the transformation from one pose to another. In the end, we will solve $AX = XB$, which in out program is *daniilidis* call. The final result X represents the transformation from camera to palm.

3.1.2 Moveit!



```
// fetch grasp point
if (gGraspPoints.size() < 1) {
    ROS_WARN("No objects detected, exit");
    break;
}
center = gGraspPoints[0];
ROS_INFO_STREAM("Object center detected: " << center);

// cut outliers
if (center.z > 0.8 || center.z < 0.05) {
    ROS_INFO("Object out of range.");
    break;
}

// Plan
group.setPositionTarget(center.x, center.y, center.z);
success = group.plan(my_plan);

// Move
if (!OnlyPlan) {
    ROS_INFO("Moving the arm to object...");
    group.move();
}
```

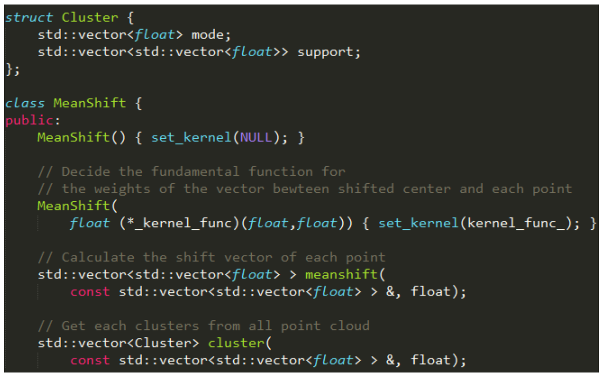
Figure 2. Plan and Move

The above code snippet shows the framework of moving the arm. First, *gGraspPoints* stores centres for all object. We fetch the first center and check its validation, i.e., whether it is empty or is lying outside or below the table plane. Second, we set position target of the end effector to that point by calling *setPositionTarget()* and *plan()* . Finally, if we are confident to go, *OnlyPlan* is set to *false* and the trajectory controller will move the arm by calling *move()*.

3.1.3 Mean Shift

The mean shift algorithm for getting each cluster contains three main parts:

Firstly, we should build Class MeanShift by input the basis-function or use the default function. Secondly, Function meanshift is to calculate the shift vector between the current center and each point based on a basis-function which measures this vector. In our experiment, we take Gaussian as the basis-function. We keep shifting each center until the shift vector below the threshold which depends on the accurate we want. The more accurate we want, the longer time this algorithm will take. Finally, Function cluster is to judge the cluster that each point belong to. These decision base on the clusters flags which are taken by these points from the Function meanshift.



```
struct Cluster {
    std::vector<float> mode;
    std::vector<std::vector<float>> support;
};

class MeanShift {
public:
    MeanShift() { set_kernel(NULL); }

    // Decide the fundamental function for
    // the weights of the vector bewteen shifted center and each point
    MeanShift(
        float (*_kernel_func)(float,float)) { set_kernel(kernel_func_); }

    // Calculate the shift vector of each point
    std::vector<std::vector<float> > meanshift(
        const std::vector<std::vector<float> > &, float);

    // Get each clusters from all point cloud
    std::vector<Cluster> cluster(
        const std::vector<std::vector<float> > &, float);
```

Figure 3. Functions of Mean Shift Algorithm

3.2. Human Interaction

Human interaction is a interesting tool to play with. The robotic arm is able to mimic human hands, i.e., move the same as human hands move.

This function is implemented using a leap motion. Leap motion is a device used for tracking finger and palm translation. First, we align the coordinate system of leap motion with that of the robotics arm base joint. Then, human arm translation under the frame of leap motion is transformed to that under the frame of base joint. Finally, this translation is interpreted by **Moveit!** and, afterwards, sent to the robotic arm. Due to the alignment of both coordinate systems, the movements of robotic is the same as that of human arm.

4. System Evaluation

One of our goals is to grasp multiple objects as quickly and correctly as possible. To define ”quickly”, we come up with a measure: ”touch 2 objects in 5 minutes, 3 objects in 10 minutes”. The reason why we want to ’touching’ instead of ’picking up’ is that we have difficulties controlling fingers of the robotic arm. In experiments, we have implemented two vision algorithms: mean shift and blob. When using mean shift, time will increase exponentially as objects increases. While using blob, the detection can be done in several seconds.

Another goal of our project is that, as human arm moves, the robotic arm should perform the same movement. This imitation should be done almost in real time. In our experiments, time interval differs. When a short path is planed, the movement can be done in one or two seconds. but when a complex path is planed, the behaviour of the robotic arm will be strange.

5. Conclusion

In this project, we proposed a programmable system that can control robotic arm to grasp objects. Till now, we are able to detect multiple objects and calculating their positions. Further more, we are able to move the arm to those positions. Human interaction is involved in our project, allowing the robotic arm to mimic the movements of a human hands.

References

[1] B. Tondu, S. Ippolito, J. Guiochet, and A. Daidie, "A seven-degrees-of-freedom robot-arm driven by pneumatic artificial muscles for humanoid robots," *The International Journal of Robotics Research*, vol. 24, no. 4, pp. 257–274, 2005.

[2] K. Kosuge, K. Furuta, and T. Yokoyama, "Virtual internal model following control of robot arms," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4, pp. 1549–1554, IEEE, 1987.

[3] K.-M. Lee and D. K. Shah, "Kinematic analysis of a three-degrees-of-freedom in-parallel actuated manipulator," *IEEE Journal on Robotics and Automation*, vol. 4, no. 3, pp. 354–360, 1988.

[4] D. Bassily, C. Georgoulas, J. Guettler, T. Linner, and T. Bock, "Intuitive and adaptive robotic arm manipulation using the leap motion controller," in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pp. 1–7, VDE, 2014.

[5] A. Saxena, J. Driemeyer, J. Kearns, and A. Y. Ng, "Robotic grasping of novel objects," in *Advances in neural information processing systems*, pp. 1209–1216, 2006.

[6] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *arXiv preprint arXiv:1603.02199*, 2016.

[7] M. E. Moran, "Evolution of robotic arms," *Journal of robotic surgery*, vol. 1, no. 2, pp. 103–111, 2007.

[8] M. Karbasi, Z. Bhatti, P. Nooralishahi, A. Shah, and S. M. R. Mazloomnezhad, "Real-time hands detection in depth image by using distance with kinect camera," *International Journal of Internet of Things*, vol. 4, no. 1A, pp. 1–6, 2015.

[9] S. Rosa, A. Russo, A. Saglimbeni, and G. Toscana, "Vocal interaction with a 7-dof robotic arm for object detection, learning and grasping," in *The Eleventh ACM/IEEE International Conference on Human Robot Interation*, pp. 505–506, IEEE Press, 2016.

[10] O. Kroemer, R. Detry, J. Piater, and J. Peters, "Active learning using mean shift optimization for robot grasping," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2610–2615, IEEE, 2009.

[11] F. Rigual Aparici, "Object recognition applied to mobile robotics," 2012.

[12] "schunk canopen driver ros package." `http://wiki.ros.org/schunk_canopen_driver`.

[13] "leap motion ros package." `http://wiki.ros.org/leapmotion`.

[14] "openni2 launch ros package." `http://wiki.ros.org/openni2_launch`.