Preliminary Implementation of Grasping Operation by a Collaborative Robot Arm: Using a Ball as Example

Wen-Chang Cheng[a], Chien-Hung Lin[a], Cheng-Yi Shi[a], Hung-Chou Hsiao[b], Chun-Lung Chang[c]
[a]*Department of Computer Science & Information Engineering Chaoyang University of Technology*, Taichung, Taiwan
[b]*Department of Information Management Chaoyang University of Technology*, Taichung, Taiwan
[c]ITRI, Hsinchu, Taiwan
{wccheng[a], s10727113[a], s10727119[a], s10814902[b]}@cyut.edu.tw, [c]vincentchang@itri.org.tw

***Abstract*—Grasping objects is one of the basic functions of a robot arm. This study completed the implementation of the process in which a collaborative robot (cobot) arm grasps an object. Hardware components included a depth camera, cobot arm, and artificial intelligence equipment for edge computing. Software components included computer visualization techniques, deep learning, and robot operating system. To complete the preliminary implementation of the system, the grasping operation of the robot arm was set to target a ball. This system implementation sheds light on how robot arms and deep learning techniques are applied to real-life problems. Experiments verified that the preliminary system implemented was able to correctly complete the ball-grasping operation and achieve the pragmatic goal.**

***Keywords*—Edge computing, computer vision, robot operating system, deep learning, object detection.**

I. Introduction

Robot arm has been successfully and widely applied in industrial manufacturing. Its scope of application has further expanded in recent years to include health care, service sector, long-term care, and agriculture. Irrespective of application, grasping objects is the most fundamental function of robot arms. For humans, grasping objects with their arm is easy, but for robots, this action is difficult. Robot arm operations generally include object detection, grasping point detection, and coordinate transformation.

Object detection involves using system-based computer visualization techniques to locate the object to be picked up. Conventional methods of object detection entail image processing, whereas advanced methods of object detection are based on a deep learning model. In this study, the deep learning-based method of object detection was employed. Grasping point detection determines the location coordinate of the target object for which a robot arm is aiming and the pose of the robot arm grasping the object. For simpler system implementation, the target object was a ball in this study, a robot arm claw of suitable size was selected, and the grasping point of the ball was the center of the ball. Object detection results can be used to calculate the center of the circle (ball); however, the circle center is expressed as two-dimensional (2D) image coordinate. Therefore, a depth camera was used to convert 2D image coordinates into coordinates in a three-dimensional (3D) space. Moreover, a specific pose for grasping the ball is not required, thus making it easier to implement the object grasping operation.

Numerous applications of high-level intelligent robots have been developed in recent years using robot operating system (ROS) [1]. ROS is a relatively new technology, first introduced in 2007. Its core concept lies in the hope that research in the fields of robotics can develop higher levels of applications without having to start from low-level software development every time. ROS is an open source platform. ROS, despite its name, is not an operating system. ROS is more of a frame responsible for handling communications and operations between every module of a robotic system. In the ROS frame, a project is called packages that contain a number of independent programs called nodes. These nodes communicate by means of a topic; nodes can publish a topic or subscribe to a topic, thereby communicating with other nodes this way. Nodes operate independently of each other so that an error with a node does not affect the operation of other nodes. Coordinate transformation has invariably been a difficult task for robots to learn. When performing a simple action, humans only require a very short time to think, carry out the action, and complete the action. By contrast, robots require an immense volume of calculation and coordinate transformation. Therefore, a transform (TF) package for ROS has been developed to address this problem. In this study, the ROS was used as the framework for system development. First, the camera calibration package in ROS was used to obtain camera intrinsic parameters, and the depth camera package in ROS was activated to publish image-related topics. Second, the YOLO package in ROS [3–7] was employed to complete object detection and publish topics on the results of object detection; the TF package in ROS was used to complete coordinate transformation and publish topics relevant to the coordinate system. Finally, the action, grasping, was completed using the ROS package provided in the cobot arm system.

The remaining sections of this paper are as follows. Section 2 introduces the system architecture; Section 3 introduces the YOLO, camera calibration, and TF packages in ROS; Section 4 presents the experimental results and discussion; and the last section concludes this paper.

II. System Architecture

The architecture of the system proposed in this study is shown in Fig. 1. The software system is based on the ROS, in which every module communicates with each other as a unit of node. The Camera SDK node publishes information on the image taken (RGB and depth); the YOLO node subscribes to the RGB image information, identifies the object, and then publishes the identified object. "Object Detect" is a self-defined node that (1) obtains information on the clicked screenshot, (2) transforms the image coordinate and camera coordinate of the identified object, and (3) publishes the camera coordinate. The TF node is a coordinate transformation package in the ROS that (1) converts camera coordinates into TF format, and (2) defines the transform of depth camera and robot arm, (3) the transform of object grasping point and object center point, (4) the transform of camera coordinates and world coordinates, and (5) the transform of robot arm and grasping point. The Kinova node reads the post-TF coordinate of the arm and activates the arm to perform the grasping action.

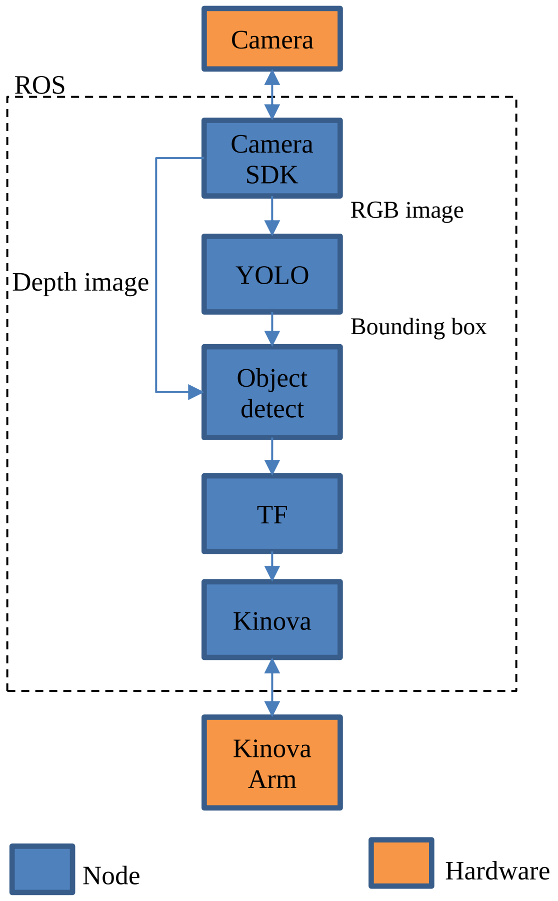


Fig. 1. System architecture

III. System Method

A. ROS–YOLO Object Detection

YOLO is a deep network model for object detection. YOLOv5 is currently available [3–7]. YOLO works by dividing an image into multiple regions and matching each block based on similarity. The higher the similarity of a match, the higher is the weight. Finally, the weights of all regions are compared to identify the region with the highest weight, and the region with the highest weight is exported. This approach is a one-stage object detection method, in which an object is located and identified simultaneously. In simple terms, a deep network model can locate and identify an object at the same time; its advantages include extremely fast processing speed, which effectively improves efficiency albeit at the expense of accuracy. Nevertheless, the accuracy of the model in identifying objects remains within an acceptable range.

Presently, YOLOv1, v2, and v3 are implemented in the ROS frame, called Darknet_ROS [8, 9]. This study used YOLOv3 on a graphical processing unit (GPU) to achieve practical application. Object_detector, bounding_boxes, and detection_image were topics published using the Darknet_ROS package. In this study, the topic, bounding_boxes, was subscribed to obtain the class name and coordinates of the detected object.

B. Camera Calibration

The main purpose of camera calibration is to obtain camera intrinsic parameters: focal length f_x and f_y and the position of the point at which the camera focal length intersects the z-axis of the image plane x-y (c_x, c_y). Camera calibration uses multiple known world coordinates and corresponding image coordinates to determine the camera intrinsic parameter matrix. In other words, 3D world coordinates are transformed into 3D camera coordinates by using the camera extrinsic parameter matrix, and the 3D camera coordinates are transformed into 2D image coordinates by projecting to the camera intrinsic parameter matrix. According to the imaging principle of a pinhole camera model, assuming that the world coordinate at Point Q is (X, Y, Z), and the image coordinate is (u, v), the following equation (1) showing the camera matrix relationship can be obtained:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{1}$$

where **K** is the intrinsic parameter matrix and [**R**|**t**] is the extrinsic parameter matrix. The camera calibration package in ROS [10] can be used to obtain the camera intrinsic parameter matrix **K**, from which the 3D camera coordinates Xc and Yc of 2D image coordinate (u, v) can be derived. Subsequently, depth camera is used to determine the 3D camera coordinate Zc of 2D image coordinate (u, v).

C. ROS-TF

TF in ROS is a package that allows for the recording of multiple coordinates at any time [2]. TF maintains the relationship between coordinate frames in a tree structure buffered in time, and allows users to transform points between any two coordinate frames at any desired point in time. TF defines the data format and data structure of the transformed coordinate. TF is essentially a tree-shaped data structure and is therefore generally referred to as TF-tree. TF can be viewed as a topic. The message of a topic is stored in the TF-tree format, thus maintaining the relationship between the coordinate transforms of the entire robot. TF package comes with a number of tools, such as tools for visualizing the transforms between nodes.

The robot model in ROS comprises a large number of parts, which are collectively called a link; each link corresponds to a coordinate frame. Link and frame are conceptually tied together and are thus processed using TF-tree in order to maintain the relationship between each coordinate frame. Any two frames must be linked, and a break in a link causes system failure, which is why the entire TF-tree cannot have any breaks. Between every two frames is a broadcaster that ensures both frames are linked correctly; a node, called TransformStamped.msg, is placed in the middle to publish messages. Therefore, a broadcaster is the publisher handling the data format between two frames, as depicted in Fig. 2.

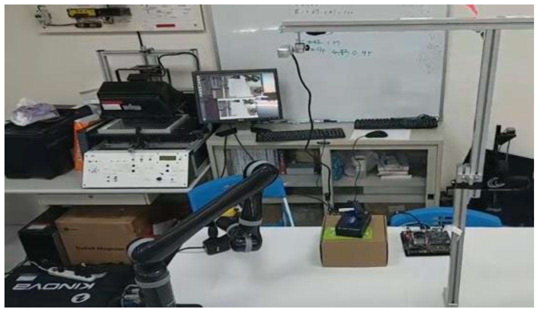
This standard formatting involves first, a header, which defines the sequence (seq), time (stamp), and name (frame_id), and then the child coordinate frame (child_frame), Transform transform and Vector 3 translation between two frames, and Quaternion rotation are written. There might be a number of nodes above a topic TF, sending messages that form a TF tree. Fig. 3 shows an example of TF-tree, in which the system defines a camera_link frame. The TF-tree comprises a camera_color_frame, which sequentially defines camera_color_optical_frame, ball_frame, and pickup_frame, and a camera_depth_frame, which defines the camera_depth_optical_frame.

```
std_mags/Header header
uint32 seq
time stamp
string frame_id
string child_frame_id
geometry_msgs/Transform transform
geometry_msgs/Vector3 translation
float64 x
float64 y
float64 z
geometry_msgs/Quaternion rotation
float64 x
float64 y
flaot64 z
float64 w
```

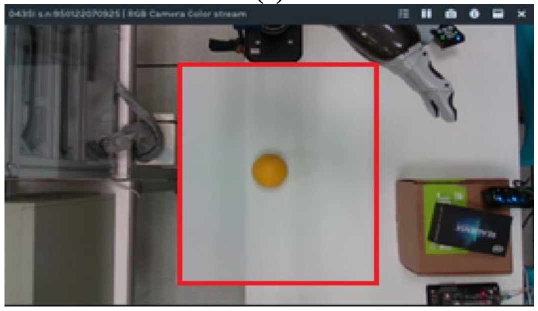
Fig. 2. TransformStamped.msg formatting

IV. Experiments and Results

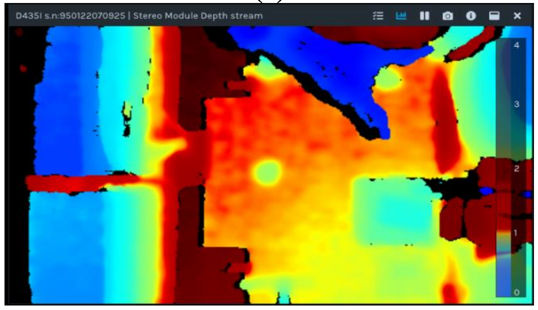
The hardware components used in this study included Jetson TX2 [11], Kinova Gen2 cobot arm [12], and a depth camera (Intel realsense d435i)[13]. Jetson TX2 is a high-performing computer for edge computing; it is primarily used to run ROS and development software programs. Kinova Gen2 is a cobot arm with six degrees of freedom (DOF) and a three-finger gripper; it supports the ROS. ROS-related packages used were Darknet-ROS, ROS camera calibration package, and ROS coordinate transform package. The implemented system is shown in Fig. 4, including the experimental setup (Fig.4(a)), camera color image input (Fig.4(b)), and depth image input (Fig.4(c)). In this study, the object to be picked up was a ball, with a recognition rate approximating 100%. The range of motion of the arm was limited to 60(L)×50(W) cm^2 as shown in Fig. 4(b). During the experiment, the depth camera was tilted at a fixed angle of 90° (optical axis vertical to desk top), which captured a small range of image but ensured greater stability. Following multiple experimental attempts to grasp the object within the range, the success rate was nearly 100%. Error had occurred during a few attempts, which upon analysis were mainly due to erroneous depth information when the depth camera was detecting the object. A part of the experimental process is available on YouTube (https://youtu.be/ulo8CwRrgbM).



(a)



(b)



(c)

Fig. 4. Implemented system; (a) experimental setup, (b) color image input and the range of grasping, and (c) depth image input.

V. Conclusion

This study used depth camera, cobot arm, and ROS-embedded platform to complete the preliminary implementation of using a cobot arm to grasp an object. For simpler grasping operation, a ball was set as the initial target. The success rate was nearly 100% according to experimental verification. Because this study involved only a preliminary implementation, the authors will in the future complete a broader range of grasping, add more target objects for grasping point detection, attempt automatic grasping of unknown objects, and incorporate multi-depth camera for improved accuracy.

Acknowledgments

This research is supported by the Ministry of Science and Technology project. The name of the project is "The implementation of robotic arm gripping objects" and the number is MOST 110-2637-E-324-001.

Reference

[1]. ROS, 2021/7/30, Online available at https://www.ros.org/.

[2]. TF-ROS, 2021/7/30, Online available at http://wiki.ros.org/tf/Tutorials.

[3]. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi,“You Only Look Once: Unified, Real-Time Object Detection”, https://arxiv.org/pdf/1506.02640v1.pdf.

[4]. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi,“You Only Look Once: Unified, Real-Time Object Detection”, https://arxiv.org/pdf/1506.02640v2.pdf

[5]. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi,“You Only Look Once: Unified, Real-Time Object Detection”, https://arxiv.org/pdf/1506.02640v3.pdf

[6]. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi,“You Only Look Once: Unified, Real-Time Object Detection”, https://arxiv.org/pdf/1506.02640v4.pdf

[7]. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi,“You Only Look Once: Unified, Real-Time Object Detection”, https://arxiv.org/pdf/1506.02640v5.pdf

[8]. Darknet-ROS, 2021/7/30, Online available at http://wiki.ros.org/darknet_ros.

[9]. Marko Bjelonic, “YOLO ROS: Real-Time Object Detection for ROS”, 2021/01/05, Online available at https://github.com/leggedrobotics/darknet_ros,

[10]. ROS.org, “How to Calibrate a Monocular Camera,” 2021/7/30, Online available at http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration.

[11]. Jetson TX2, 2021/7/30, Online available at https://www.nvidia.com/zh-tw/autonomous-machines/embedded-systems/jetson-tx2/.

[12]. kinova gen2, 2021/7/30, Online available at https://www.kinovarobotics.com/en/products/gen2-robot(accessed on 2020/05/25)

[13]. Intel realsense d435i, 2021/7/30, Online available at https://www.intelrealsense.com/depth-camera-d435/

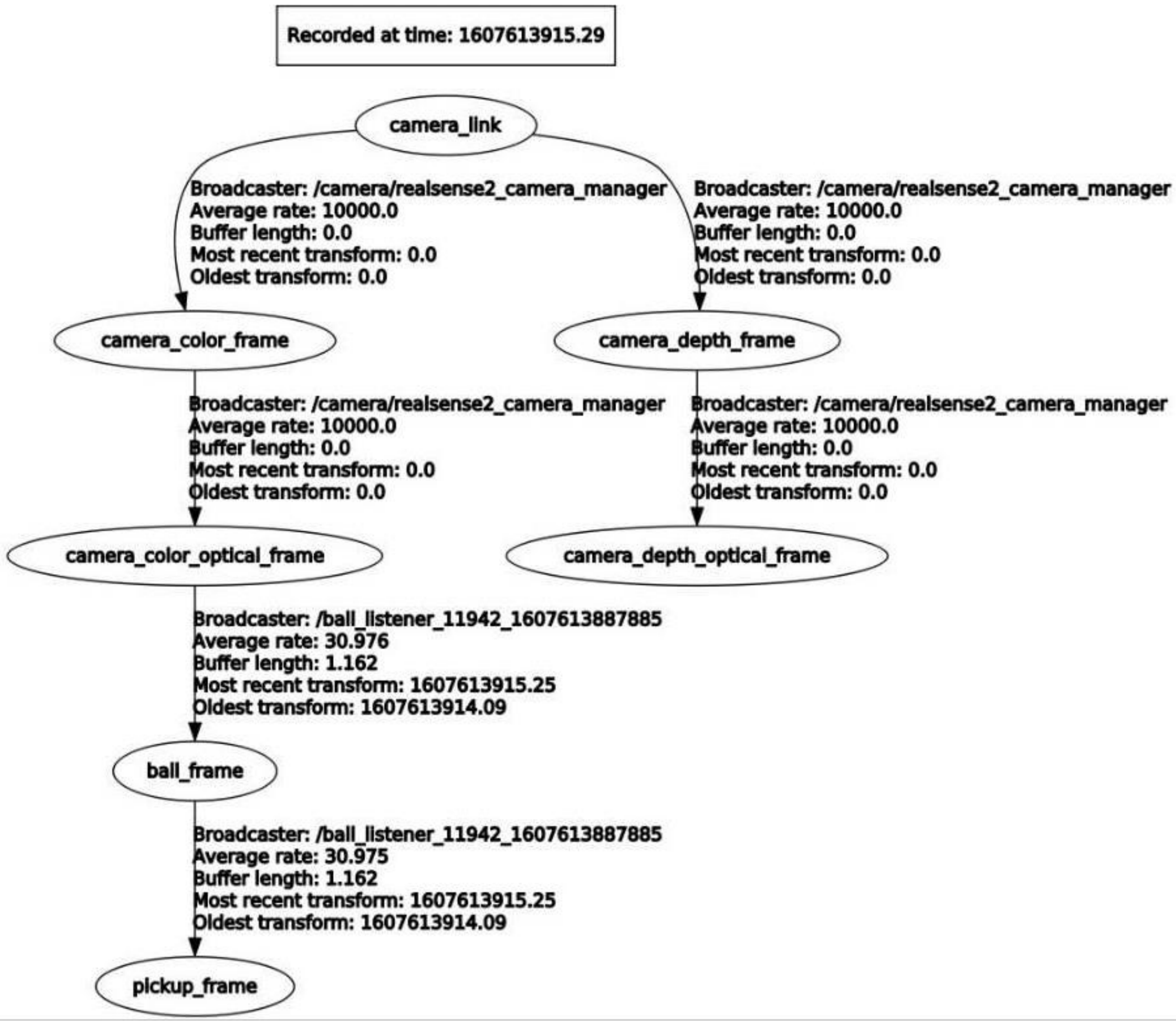

Recorded at time: 1607613915.29
camera_link
Broadcaster: /camera/realsense2_camera_manager
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /camera/realsense2_camera_manager
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_color_frame
camera_depth_frame
Broadcaster: /camera/realsense2_camera_manager
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /camera/realsense2_camera_manager
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_color_optical_frame
camera_depth_optical_frame
Broadcaster: /ball_listener_11942_1607613887885
Average rate: 30.976
Buffer length: 1.162
Most recent transform: 1607613915.25
Oldest transform: 1607613914.09
ball_frame
Broadcaster: /ball_listener_11942_1607613887885
Average rate: 30.975
Buffer length: 1.162
Most recent transform: 1607613915.25
Oldest transform: 1607613914.09
pickup_frame

Fig. 3. Camera coordinate and ball coordinate systems