MASTER THESIS

Lada Kudláčková

Manipulating Objects through Deictic Gesture Recognition

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: RNDr. David Obdržálek, Ph.D.
Study programme: Computer Science

Prague 2024

Dedication.

Title: Manipulating Objects through Deictic Gesture Recognition

Author: Lada Kudláčková

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Use the most precise, shortest sentences that state what problem the thesis addresses, how it is approached, pinpoint the exact result achieved, and describe the applications and significance of the results. Highlight anything novel that was discovered or improved by the thesis. Maximum length is 200 words, but try to fit into 120. Abstracts are often used for deciding if a reviewer will be suitable for the thesis; a well-written abstract thus increases the probability of getting a reviewer who will like the thesis.

Keywords: gesture recognition, object manipulation, autonomous control

Název práce: Manipulace s objekty pomocí rozpoznávání ukazovacích gest

Autor: Lada Kudláčková

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. David Obdržálek, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Abstrakt práce přeložte také do češtiny.

Klíčová slova: rozpoznání gest, manipulace s objekty, autonomní řízení

Contents

Introduction

TODO:

Opening statement: introducing the research field, stating the problem.
My motivation, goals and research limitation.
Overview of the thesis structure.

1 Task Analysis

1.1 Basic Terminology

1.1.1 Gesture Recognition

Gesture recognition technology uses a computer and a sensor to interpret human body movements. It allows direct control of machines without mechanical devices such as a keyboard or joystick.

Body movements are captured by a sensor. A static gesture can be identified in a single frame of raw sensor data, dynamic has to be tracked in consecutive frames during the movement.

The gesture is assigned to one of the predefined gesture types and the corresponding gesture type is translated into machine commands.

The gesture recognition process can be divided into the following parts:

- sensor data collection
- gesture identification
- gesture tracking
- gesture classification
- gesture mapping

TODO: ref
Hongyi Liu, Lihui Wang (2017). Gesture recognition for human-robot collaboration: A review.
International Journal of Industrial Ergonomics, Volume 68, November 2018 (355-367). https://doi.org/10.1016/j.ergon.2017.02.004

1.1.2 Deictic Gesture

A deictic gesture is a gesture that indicates direction or location from the perspective of the person performing the gesture.

The meaning of the deictic gesture depends on the context, it can refer to a real or a virtual environment. It could often be expressed by adverbs such as "here" and "there" or by demonstrative pronouns such as "this" and "that". We use it to specify direction and location or to identify a person or an object in the environment.

The most common deictic gesture is the pointing gesture. Other examples are gestures based on head movements or eye gaze.

1.1.3 Pointing Gesture

A pointing gesture is performed by extending the arm in the appropriate direction, usually using the index finger or hand to indicate the direction.

It may represent the pointing ray, which is given by, for example, the eyes (as the origin) and the index finger, or it may have a more symbolic meaning, such as when a person is pointing outside his field of vision or in a virtual environment.

Pointing with the index finger is a cross-cultural behavior. Infants most commonly use their index fingers for tactile exploration of their environment and they often use the gesture of the extended index finger for a variety of purposes before they acquire its social meaning.

1.1.4 Object Manipulation

Object manipulation in robotics refers to robot's interaction with its environment that involves physical contact with an object and causes a change in the position, orientation, or shape of the object.

There are various ways a robot can move or modify an object. Industrial robots typically grasp objects using a robotic arm with a gripper or other end effector. Robotic lawn mowers use rotating blades to cut and generate an air flow to collect grass.

In Robot Sumo, robots use a blade to push their opponents out of the arena. Humanoid robots perform actions that are more similar to human actions: they can kick a ball with their legs or use a golf club to hit the ball.

Simple manipulation activities can be performed using a predefined set of commands, but more complex tasks require the robot to plan and control the motion.

1.1.5 Autonomous Control

Autonomous control is the ability of a machine to perform tasks independently, without direct human intervention. There are different degrees of autonomy.

An automated machine has no autonomy, it strictly follows human instructions by executing predefined commands and making all decisions according to predefined logic. Operates within a known framework and needs human intervention in the case of an unexpected event.

A fully autonomous machine could theoretically accomplish all its tasks without human intervention. It would use artificial intelligence for planning and control, all unexpected events could be handled by the machine itself.

In the real world, machines that are considered fully autonomous are designed to operate only in a simple, predictable environment. They still require some kind of human supervision and the supervisor can take control in an emergency.

The semi-autonomous machine lies between these two extremes. Most of the time it preforms the task independently, but in some parts of the process a supervisor is involved in decision making or direct control of the machine.

1.1.6 Human-Robot Interaction

Human-robot interaction (HRI) is an interdisciplinary field of research that studies the ways of communication between human and robots.

HRI integrates knowledge of robotics, artificial intelligence, natural language processing, engineering and psychology.

The main goal is to develop robots that behave in natural way and are able to effectively communicate with users.

This field is related to human-computer interaction (HCI), but HCI focuses primarily on software and interfaces. Interaction with robots is more physical than with computers. Robots can move around, sensors allow them to explore the environment or to learn while interacting with humans.

The HRI system is designed not only to be convenient for the user, but is also considered from the robots' point of view.

It is important to avoid robot's collisions with users as they can lead to injuries or material damage. Safety should be always ensured for both humans and robots during the whole interaction.

1.2 Gesture Based Control for Pick and Place Task

"Pick and Place" is one of the basic tasks of object manipulation: the robot's goal is to move an object to a given target location.

The robot has to be able to move around the environment, find its way to the desired locations and manipulate the object, for example, by grasping and releasing it. Object detection is necessary unless the environment is very simple and the objects have a predictable location.

In order to accomplish the task using gesture-based control, we need to execute the following steps:

Pick:

- User performs gestures to select an object.
- Robot navigates close to the selected object.
- Robot identifies the object and its exact coordinates.
- Robot picks up the object.

Place:

- User performs gestures to specify the target location.
- Robot navigates close to the target location.
- Robot places the object to the target location.

1.3 Task Specification

I designed a gesture based control for the Pick And Place task and implemented it in C++ and Python using the Robot Operating System (ROS).

Image data for object detection and gesture recognition was captured by a depth camera. The implementation was tested with a mobile robotic manipulator available in the robotics lab.

The experiments were conducted indoors. The environment was static, with only the person performing the gestures and the robot moving in the scene.

The distance from the camera in which the person could be detected was limited. I also restricted the area in front of the camera where objects could be initially located. Otherwise, objects not related to the experiment would have been detected.

A similar restriction applied to the area where the robot could work. No obstacles were placed there except for the detected objects. The target position of the selected object has to be chosen within this area.

1.4 Goals

The main goal was to implement gesture control using the devices available in the robotics lab.

The implementation should provide several different types of gestures that would be compared for accuracy. The metric for comparison is the average distance between the correct coordinates (of the selected object or location) and the coordinates determined by the gesture. The user should be able to select the gesture type.

I aimed also to demonstrate the designed control with a real mobile robotic manipulator. This objective includes designing the robotic system and the implementation of the interfaces necessary to control the movements of the mobile robot, the robotic arm and the gripper. The robot should move safely, avoid collisions and manipulate objects without unnecessary emergency braking.

2 The state of the art

2.1 History of Gesture Based Control

2.1.1 Computer Vision

The first digital image scanner was built in 1957 by Russell A. Kirsch. The device optically scanned a scene and converted the scan into an image, which was represented by pixels.

In the 1960s, David Hubel and Torsten Wiesel conducted experiments with cats that provided insight into image processing in the brain. They showed cats various simple visual stimuli while recording the electrical activity of cells in the visual cortex.

This revealed how the brain builds complex visual representations from simple elements and how individual neurons are involved in image processing, e.g. neurons responsible for edge detection were discovered.

In 1966 was founded the Summer Vision Project on MIT, which focused on machine vision and pattern recognition. The goal was to implement a visual system that would solve tasks such as distinguishing between foreground and background or extracting distinct objects.

Edge detection was implemented in 1987 using the gradient calculation.

The "Eigenfaces" face recognition algorithm was developed in 1991, the Scale-Invariant Feature Transform (SIFT) algorithm for local features detection in 1999, and the Viola-Jones face detection algorithm in 2001.

In 2005, the Histogram of Oriented Gradients (HOG) algorithm was introduced that enabled efficient body recognition and object detection.

Deep learning has become the dominant computer vision method in the following years, especially after the success of the ImageNet Challenge in 2012.

2.1.2 Gesture Recognition

In the 1960s, the first touch screens and pointing devices were developed. In 1963, Sketchpad was written, a computer program that allowed interaction with objects on the screen using a light pen to capture motion.

In 1977, Sayre's Glove was introduced, which used a light sensor and a flexible light source tube to determine the position of the fingers. Later, various sensors such as accelerometers or magnetic sensors were used for gesture recognition using gloves.

Image processing tools enabled vision-based hand gesture recognition using images of gestures performed with a color marked glove.

Body recognition methods included wearable sensors such as electromyographic (EMG) sensors attached to the arm or suit with an IMU and barometer.

Sensors such as radar enabled body recognition at a distance.

Significant was the development of depth cameras that use time-of-flight or structured light-based technologies, such as the Microsoft Kinect device launched in 2010, which enabled real-time body and gesture recognition.

2.2 Localization and navigation with deictic gestures

These are some (not all) examples of what I want to mention here:

Deictic gestures for multi-robot systems

Paper:
B. Gromov, L. M. Gambardella and G. A. Di Caro, "Wearable multi-modal interface for human multi-robot interaction," 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Lausanne, Switzerland, 2016, pp. 240-245, doi: 10.1109/SSRR.2016.7784305.

Use of the pointing gesture for localization

Paper:
B. Gromov, L. Gambardella, and A. Giusti. Robot Identification and Localization with Pointing Gestures. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 3921–3928 https://people.idsia.ch/ gromov/repository/gromov2018robot.pdf

3D Motion planning with pointing gestures

Paper:
B. Gromov, J. Guzzi, L. Gambardella, and A. Giusti. Intuitive 3D Control of a Quadrotor in User Proximity with Pointing Gestures. IEEE International Conference on Robotics and Automation (ICRA), 2020 https://people.idsia.ch/ gromov/repository/gromov2020intuitive.pdf

2.3 Interpretation of gestures

Papers:

Chaudhary, A (2018). Robust Hand Gesture Recognition for Robotic Hand Control. Springer. ISBN 978-981-10-4798-5 https://doi.org/10.1007/978-981-10-4798-5

Alikhani, M., Khalid, B., Shome, R., Mitash, C., Bekris, K.E., Stone, M. (2020). That and There: Judging the Intent of Pointing Actions with Robotic Arms. AAAI.https://ojs.aaai.org//index.php/AAAI/article/view/6601

2.4 Object detection with pointing gestures and speech recognition

Li-Heng Lin, Yuchen Cui, Yilun Hao, Fei Xia, Dorsa Sadigh (2023). Gesture-Informed Robot Assistance via Foundation Models. https://arxiv.org/abs/2309.02721

A. Ekrekli, A. Angleraud, G. Sharma, R. Pieters (2023). Co-speech gestures for human-robot collaboration. https://arxiv.org/abs/2311.18285

3 Gesture Based Robot Control

3.1 Overview

Two gestures are required to successfully perform the "Pick And Place" task: a deictic gesture to indicate the position and another gesture for confirmation.

During the selection of the object and the target location, the user should face the camera, with all objects lying on the floor between him and the camera. The scene is displayed on the rViz screen. Once the objects are detected by the vision system, their images are marked in blue.

The user can then initiate the selection of an object by pointing at it with his right hand. To confirm the gesture, he raises his left hand while still pointing at the object.

The object closest to the intersection of the pointing ray and the floor is selected. Its image is marked in red.

We can then specify the target location of the selected object. The user determines the location in the same way as before: by pointing to the location and raising his hand.

The target location is on the floor and has to be selected inside the safety frame that is shown in the rViz. The frame represents a space that is safe for the robot to move around, there are no obstacles except for the detected objects.

Once the target location is selected, it is marked in red in rViz, gesture detection is completed and the resulting data is sent to the robot.

3.2 Gestures

3.2.1 Pointing Gesture

The user can choose from three types of pointing gestures. Each type is represented by a pair of joints that corresponds to the pointing ray. The first joint in the pair is the origin and the second determines the direction of the ray:

- Shoulder, wrist (default option).
- Elbow, wrist.
- Head, hand.

The pointing gesture indicates the point where the pointing ray intersects the floor and allows us to select an object or its target location.

The pointing ray is displayed from its origin to the intersection with the floor.

The pointing gesture has to be performed with the right hand and the first joint has to be positioned higher than the second. These constraints help to reduce the number of falsely detected gestures.

3.2.2 Raising Hand Gesture

The hand gesture consists of lifting the hand. It has to be performed with the left arm and the hand has to be raised above the head.

When pointing with the right arm, the user confirms the pointing gesture by raising the left hand. If no pointing gesture is performed in the moment, the raising hand gesture is ignored.

3.3 Gesture Recognition with ORBBEC Astra SDK

3.3.1 Limitations

ORBBEC Astra SDK provides tools for skeleton recognition and person tracking. The maximum distance for skeleton recognition is 4 meters. Multiple persons can be tracked at the same time.

The skeleton is represented by a set of joints and their positions. The head corresponds to single joint, eye positions and other details are unavailable.

Three joints are given for each arm: shoulder, elbow and hand. The fingers are not recognizable.

SDK also supplies the detection of the grip gesture. I considered using it as a confirmation gesture but preferred the hand raising gesture because the grip was often not detected.

3.3.2 Occlusion

The most common cause of gesture recognition errors is occlusion. Only relatively small objects (not taller than 20 cm) were used for the experiments. If the objects were larger, many false detections occurred because the objects often obscured parts of the person's body and the joints of the corresponding skeleton were not correctly identified.

Occlusion also often occurs when more than one person is in front of the camera. Therefore, it is better to perform the Pick And Place task when only one person is in the scene because it makes gesture recognition more robust.

However, if more than one person is present, each person can perform a pointing gesture to select an object or a target location.

3.3.3 Recommendations

When I tested skeleton recognition with a single person and small objects, most of the errors were caused by the person's posture.

For best results, the person should be facing the camera, not turning the body or crossing his limbs, as this can lead to errors such as interrupted tracking of the person, misidentification of joints and false recognition of gestures.

In case of difficulties with skeleton recognition, it may help if the person moves closer to the camera or extends his arms out to the sides.

4 Design of Robotic System

4.1 Basic Structure

The proposed robotic system contains two main components: a vision system and a mobile robotic manipulator. The vision system is static, camera remains at the designated location during the task.

The robot starts at the initial position where it waits for messages from the vision system. Once the object and target position are selected and a result message is received, the robot navigates to the object, moves it to the target position and returns to the initial position.

4.1.1 Vision System

The main purpose of the vision system is to interpret the environment: to detect objects and the skeleton of a person with its gestures. The position of the objects and the tracked person is limited - outside a given frame, the detection is unreliable.

For the ORBBEC Astra, the distance must be less than 4 meters. Therefore, I decided to use a vision system separate from the mobile robotic manipulator. Otherwise, with a camera attached to the robot, only objects very close to it could be detected.

The depth camera is connected to a laptop where the input data from the camera is processed. The resulting message is sent to the robot's desktop computer.

4.1.2 Mobile Robot Manipulator

The robot consists of a mobile vehicle with a robotic arm and a gripper.

The vehicle is equipped with a laser scanner that enables localization and safe autonomous navigation in the environment. The on-board computer serves as the robot's ROS master and is connected to a desktop computer.

The arm with gripper is attached to the vehicle. It needs to be set up so that there are no collisions with the robot or the floor during manipulation. The reach of the arm should also be limited so that it does not move within the field of view of the scanner, as this would trigger an emergency stop.

The arm computer controls both the arm and the gripper and is available via ROS and Dashboard Server.

4.2 Hardware

4.2.1 Neobotix MP-500

The MP-500 mobile robot is a differential-wheeled robot with two large drive wheels and one small one at the rear. It is one of the most robust Neobotix mobile robots with a weight of 70 kg.

Its main components are a mobile platform, laser scanner, on-board computer, battery pack, manual charger and wireless joystick. Additional components can be attached to the mobile platform.

The robot can be used for material transport, with a load capacity of 80 kg. It is designed for indoor operation and can move with a speed up to 1.5 m/s.

A Sick S300 safety laser scanner with a maximum range of 30 meters is mounted in the front of the mobile platform. The scanner provides data that is used for localization, navigation and collision avoidance.

Detection of a person or obstacle in the safety field immediately triggers an emergency stop.

4.2.2 Robotic Arm UR5

The Universal Robots UR5 manipulator consists of a robotic arm, a control box with a teaching pendant and a battery.

The six-axis arm is composed of extruded aluminum tubes and rotational joints (Base, Arm, Elbow, Wrist 1, Wrist 2, Wrist 3). The Base is the first joint of the kinematic chain in which the arm is mounted to a fixed surface or a mobile platform. The last joint to which the tool is attached is Wrist 3.

All joints have a motion range of 360 degrees. The reach of the arm is 0,85 m from the center of the base, the area directly above and below the base is out of reach. The weight is 18.4 kg and the maximum payload is 5 kg.

The teaching pendant provides a GUI for control of the arm, commands can also be sent remotely using dedicated ports.

4.2.3 Weiss Robotics GRIPKIT

A two-finger gripper is connected to the UR5 arm using the Weiss Robotics GRIPKIT module. Its maximal opening stroke is ? TODO

4.2.4 ORBBEC Astra camera

TODO

4.2.5 Computers and network

For the vision system, an Acer TravelMate P214 notebook is used. The ORBBEC Astra camera is connected via USB.

A Lenovo ThinkStation P330 desktop computer controls the mobile manipulator. It is connected to the Neobotix MP-500 mobile robot via an Ethernet cable.

The connection between the computers is established via WiFi, messages are sent using SSH.

5 Implementation

5.1 Vision system

5.1.1 Image Processing with ORBBEC Astra Camera

The image data is sent from the camera to the connected notebook for processing.

There is the ROS Master and several other individual ROS nodes running on the notebook. Some ROS nodes are involved in image processing, while others provide tools such as geometric calculations or displaying detected objects and skeletons on the rViz screen.

ROS nodes communicate with each other using ROS messages.

I used two main tools to process the camera data: the ROS Astra camera driver ros_astra_camera for object detection and the ORBBEC Astra SDK for skeleton detection.

Both tools use OpenNI as an intermediate layer to access the camera data. I couldn't run them at the same time because it led to runtime errors, so I decided to split the process into two separate phases.

First, the ros_astra_camera driver is started. Once all objects are detected, the driver stops.

In the second phase, data is exposed by the ORBBEC Astra SDK until both pointing gestures are confirmed and the result is sent to the robot.

Since gesture recognition is performed within the ROS system, additional ROS packages were needed to publish the body tracking data provided by the SDK as ROS messages.

5.1.2 Installation

ROS Noetic

The notebook with Ubuntu 20.04 was used, for which the recommended version of the ROS distribution is ROS Noetic. I followed the instructions from http://wiki.ros.org/noetic/Installation/Ubuntu to download and install the ROS Noetic package.

Astra and OpenNI SDKs

For the ORBBEC Astra camera, I needed to install the Astra SDK and the OpenNI SDK for Linux.

Both SDKs are available at https://www.orbbec.com/developers.

ROS Driver for Astra camera

I downloaded the ROS driver package
from https://github.com/orbbec/ros_astra_camera and installed the dependencies according to the instructions on http://wiki.ros.org/astra_camera.

The ros_astra_camera package supports the ROS distributions Kinetic and Melodic. I needed to find and test multiple versions of the "ros-*-libuvc-*" libraries, as they were not released specifically for ROS Noetic.

This problem was already solved on the ORBBEC GitHub page, so I followed the advice and installed the missing dependencies using:

```
$ apt install ros-noetic-rgbd-launch libuvc-dev
```

I built the package with "catkin_make" command and was able to run code samples that show the camera data on the screen.

ROS Packages for Gesture Based Control

I downloaded three ROS packages from the Shinsel Robots repository on https://github.com/shinselrobots.

The "pcl_object_detection" package allows object detection in the camera data provided by the ROS Astra driver using the Point Cloud Library.

The "astra_body_tracker" and "body_tracker_msgs" packages publish body tracking data from the Astra SDK as ROS messages.

Several environment variables have to be set to indicate the paths to the AstraSDK and OpenNI subfolders.

For example, if "/home/user/AstraSDK" is the folder containing the Astra SDK and "/home/user/OpenNI-Linux-x64-2.3.0.66" is the folder containing the OpenNI SDK, the setting can be made by running these commands in the terminal:

```
$ export ASTRA_SDK=/home/user/AstraSDK
$ export ASTRA_ROOT=/home/user/AstraSDK
$ export ASTRA_SDK_INCLUDE=/home/user/AstraSDK/include
$ export ASTRA_SDK_LIB=/home/user/AstraSDK/lib
$ export OPENNI2_INCLUDE=/home/user/OpenNI-Linux-x64-2.3.0.66
$ export OPENNI2_REDIST=/home/user/OpenNI-Linux-x64-2.3.0.66/Redist
```

5.1.3 Source Code

Catkin Workspace

Catkin is the official build system for ROS. Project packages that are placed in the same catkin workspace can be built all at once.

My catkin workspace folder contains following ROS packages, all with source code written in C++:

- ros_astra_camera
- task_execution
- rviz_screen
- pcl_object_detection
- pointing_gesture

Program Overview

The main launch file is task_execution.launch. It starts the ROS Astra driver, rViz and other ROS nodes involved in the task: task_execution_node, pcl_object_detection_node and pointing_gesture_node.

The task_execution_node subscribes to ROS messages "pcl_object_detection/detected_objects" and "body_tracker/intersection". The "pcl_object_detection/detected_objects" message contains an array of coordinates of the detected objects, the "body_tracker/intersection" message contains the coordinates of the intersection of the pointing ray and the floor.

When the intersection message is received for the first time, the nearest object is calculated. The object is represented by its index in the detected object array, which is then published in the "task_execution/pointed_object_index" message.

The second received intersection message indicates the target location. Once received, the node creates a result file and writes the coordinates of all detected objects, the coordinates of the target location and the index of the selected object.

Then the node connects to the robot's computer using SSH, transfers the file there and remotely starts the robot's main program.

Object Detection with Point Cloud Library

Point Cloud Library (PCL) was used as a tool for processing the image data.

The pcl_object_detection package allows the detection of objects on a flat surface and depends on two PCL packages: pcl_ros and pcl_conversions.

The pcl_ros package bridges ROS systems and 3D applications that work with point clouds. It extends the ROS C++ Client Library to support messages with native PCL data types.

Conversions between PCL data types and ROS message types are provided by the pcl_conversions package.

The pcl_object_detection_node subscribes to the topic "/astra_camera/depth/points", which is published by the ROS Astra camera driver. The received messages represent a point cloud with no color information.

The point cloud is processed using PCL: the data is filtered with VoxelGrid and used for plane segmentation. The remaining points, which are outside the plane, are divided into clusters. Each cluster can eventually be considered as a detected object. We can specify parameters that limit the height and width to detect only a specific type of objects.

All clusters that meet the given conditions correspond to detected objects and their properties are published in the "pcl_object_detection/detected_objects" message.

Detection in Experimental Environment

Even if the scene with the objects was static, different objects were detected in consecutive depth clouds. The detection reliability decreased with the person moving in the scene.

I modified the original pcl_object_detection package from Shinsel Robots to improve the quality of object detection in my experimental environment. I also needed to implement a mechanism for switching between the ROS Astra driver and the Astra SDK to avoid issues with camera data accessibility.

The object size is limited to 20 cm and the object can be detected only when lying on the floor. Furthermore, a detection frame has been specified as a constraint for the object's location on the floor. An object outside the detection frame is ignored.

To ensure that the same objects are detected when the experiment is repeated in the same scene, I fixed the total number of objects and modified the program:

The received point clouds are processed one by one, with some objects detected in each cloud.

If the number of detected objects does not equal the total number, the result is ignored and processing continues with the next cloud.

Otherwise, the number of detected objects is correct, which usually means that only the correct objects have been detected. Their data is exposed in the "pcl_object_detection/detected_objects" message. In addition, the message "object_detection_done" is published, which indicates that the ROS Astra driver is no longer needed.

I added the "object_detection_done" subscriber to the ROS Astra driver. Once the message is received, the driver is shut down.

The package was also extended by code for displaying detected objects in rViz, marking them in blue and changing the color to red if the object was selected by the pointing gesture.

Gesture recognition

The pointing_gesture package implements methods for gesture recognition. It uses the Astra SDK to obtain body tracking information and allows several types of gesture recognition.

I modified the code from the astra_body_tracker package from Shinsels Robots. The package publishes body tracking information from the Astra SDK in the ROS topic, using messages from the body_tracker_msgs package.

The Astra SDK provides tools for skeleton recognition and body tracking. The tracked person is represented by a list of nineteen joints.

Corresponding body data includes the position and status of each joint, where status indicates the confidence level of the tracking with possible values of "NotTracked", "LowConfidence" and "Tracked". The data also contains the current body tracking status, body orientation and hand pose.

The pointing_gesture node is inactive during object detection. The "object_detection_done" message triggers the shutting down of the ROS Astra driver.

I added a subscriber for the "object_detection_done" topic to the pointing gesture node as well. Once the message is received, the node attempts to open the Astra data stream using the Astra SDK and repeats this until the ROS Astra device stream is completely terminated.

Then the data stream is available for the Astra SDK and we can start gesture recognition.

The Astra SDK provides a list of tracked persons for each frame of the image data stream. For each tracked person, a check is made to evaluate whether he is performing the confirmation and pointing gestures.

The confirmation gesture is performed by raising the left hand. It is detected when the left hand joint is raised above the head joint.

I tried using the shoulder joint instead of the head joint. However, body tracking is not completely reliable and I observed many cases of false gesture detection. The shoulder joint has been often misidentified, while the head joint is usually identified correctly.

If a confirmation gesture is detected in a given frame, the right arm is checked for the pointing gesture, otherwise processing continues with the next frame.

The pointing gesture is recognized by the position of the pair of joints corresponding to the selected pointing gesture type. The first joint of the pair determines the origin of the pointing ray. It is the upper joint when we consider the standard anatomical position.

The pointing gesture is detected when the first joint is positioned higher than the second.

The coordinates of the pointing ray intersection with the floor plane are computed and published in a ROS message.

The default option for the pointing gesture is the head-wrist type. The user may select a different option using a ROS message (TODO: add command example.

I have added rViz markers to display the skeleton of the tracked person and the pointing ray. When the first confirmed pointing gesture is detected and the intersection coordinates are published, the object closest to the intersection is selected and marked red in rViz. The distance between the intersection point and the object is logged.

The second detected pointing gesture specifies the target location. The coordinates of the corresponding intersection are also published and marked red in rViz.

Once the second confirmed pointing gesture is detected and the intersection is published, the data stream is closed.

5.2 Mobile Robot Manipulator

5.2.1 Execution of Pick And Place Task

I aimed to implement a Pick And Place task to test gesture based control with a real robot system and decided to use a mobile robotic manipulator for experiments.

Not all features for autonomous execution were fully implemented. I simplified the task because both navigation and object detection require detailed analysis and testing.

There is a restricted area in the robotics lab for the robot, with no obstacles except a few small objects on the floor. The surface is flat and even.

The robot knows its initial position and the coordinates of the camera. Once it receives the coordinates of the selected object and its target location, it navigates to the selected object and extends the robotic arm forward.

Since I have not completed the autonomous control of the robot arm, the gripper is not placed in the correct position and the robot needs help to pick up the object.

After five seconds, it closes the jaws of the gripper and navigates to approximate target location where it releases the object.

The robot cannot handle unexpected events and will stop in a case of emergency.

5.2.2 Autonomous Navigation

The Neobotix Mobile Robot

The ROS Kinetic and Neobotix ROS packages have already been installed on the Neobotix mobile robot computer, as well as the ROS packages for navigation and motion control (ros-noetic-amcl, ros-noetic-map-server, ros-noetic-move-base).

I installed the ROS Noetic and packages from the Neobotix GitHub repository on my notebook.

The move_base framework provides access to the ROS navigation stack and moves the robot to its goal destination. I implemented a SimpleActionClient that sends the goal state to move_base.

The goal state includes the position and orientation of the robot. I needed to tune the parameters of the local Neobotix planner: the tolerance for the target orientation had to be increased because it was too strict for the robot.

I also modified robot's behaviour when unknown obstacles are detected.

When the laser scanner detects an unknown obstacle in front of the robot, the robot creates a new motion plan and turns sideways to avoid a collision. This may lead to the detection of other new obstacles.

By default, the robot will start to perform a recovery rotation when it cannot find any way out of its current position. I disabled this setting and increased the threshold for obstacle distance to enable the robot to approach the objects without unnecessary turning.

Map of Environment

The environment map can be created using the neo_mp_500 package, which provides tools for autonomous navigation. Simultaneous localization and mapping (SLAM) is performed using the gmapping algorithm.

I moved the robot around the lab using the remote control, checked the mapping process using rViz, and saved the map when it adequately represented the scene.

TODO: ref https://github.com/neobotix/, add image of map and rViz screenshot

5.2.3 Object Manipulation

Universal Robots UR5 Manipulator

TODO:
Universal Robots:
DashBoardServer, Polyscope, URP, ...

Packages:
Universal_Robots_ROS_Driver https://github.com/UniversalRobots/Universal_Robots_ROS

Universal_Robots_Client_Library
https://github.com/UniversalRobots/Universal_Robots_Client_Library

ur5_moveit_config
https://github.com/ros-industrial/universal_robot/tree/noetic-devel/ur5_moveit_config

Mobile Manipulator URDF

TODO:
URDF for Neobotix, UR5 and gripper.

MoveIt Setup Assistant

How to create config and set up arm positions.
How to set up arm limits.
Simulation in rViz.

6 Experiments

TODO:
Experiments descriptions:
Experiments with different ways of using deictic gestures:
- a pointing ray calculated from a pair of skeleton coordinates (head - hand, elbow - wrist, shoulder - wrist)
- pointing with or without visual feedback (pointed ray shown in rViz)

Experiments measurements:
...

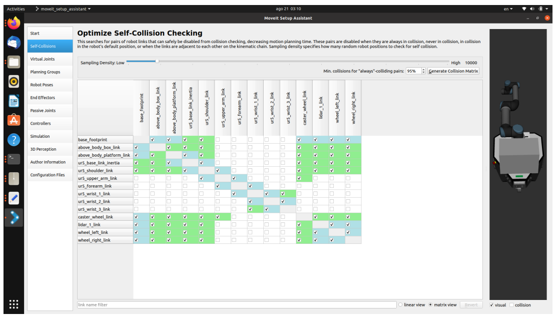
result evaluation; what went wrong; future work, possible improvements

Conclusion

Results of experiments - summary.
Which gestures are well recognised by Astra camera;
most accurate pointing gestures - compare results with related work.

Suggestions for improvement.

7 Appendix



Optimize Self-Collision Checking

This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling density specifies how many random robot positions to check for self collision.

Sampling Density: Low — High 10000

Min. collisions for "always"-colliding pairs: 95%

	base_footprint	above_body_box_link	above_body_platform_link	ur5_base_link_inertia	ur5_shoulder_link	ur5_upper_arm_link	ur5_forearm_link	ur5_wrist_1_link	ur5_wrist_2_link	ur5_wrist_3_link	caster_wheel_link	lidar_1_link	wheel_left_link	wheel_right_link
base_footprint		✓	✓	✓	✓						✓	✓	✓	✓
above_body_box_link	✓		✓	✓	✓						✓	✓	✓	✓
above_body_platform_link	✓	✓		✓	✓						✓	✓	✓	✓
ur5_base_link_inertia	✓	✓	✓		✓						✓	✓	✓	✓
ur5_shoulder_link	✓	✓	✓	✓		✓					✓	✓	✓	✓
ur5_upper_arm_link					✓		✓				✓			
ur5_forearm_link						✓		✓						
ur5_wrist_1_link							✓		✓	✓				
ur5_wrist_2_link								✓		✓				
ur5_wrist_3_link								✓	✓					
caster_wheel_link	✓	✓	✓	✓	✓	✓						✓	✓	✓
lidar_1_link	✓	✓	✓	✓	✓						✓		✓	✓
wheel_left_link	✓	✓	✓	✓	✓						✓	✓		✓
wheel_right_link	✓	✓	✓	✓	✓						✓	✓	✓	

Figure 7.1 Enter Caption

7.1 First section

Let's cite! The Einstein's journal paper [1]

Bibliography

1. STUDENT. On the probable error of the mean. *Biometrika.* 1908, vol. 6, pp. 1–25.

List of Figures

List of Tables

List of Abbreviations

A Attachments

A.1 First Attachment