Technical Report

IRI-DT 14-02

"Teaching Grasping Points Using Natural Movements"

Yalım Işleyici
Guillem Alenyà

July, 2014

Institut de Robòtica i Informàtica Industrial

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain
Tel (fax): +34 93 401 5750 (5751)
http://www.iri.upc.edu

Corresponding author:
Yalim Isleyici
tel: +34 93 405 4490
yisleyici@iri.upc.edu
http://www.iri.upc.edu/staff/yisleyici

1 Introduction

1.1 Objectives

A common strategy to teach a robot certain skills involves demonstration. While the demonstrations are best made by directly manipulating the robot, in hazardous conditions the only choice is teleoperation. Even though haptic devices offer fairly good results, using natural movements would give a better feeling of control.

Leap Motion sensor (Leap Motion Inc., USA) is a device that detects hands and it can be used to control the robot arm in a more natural way. In this work, a system that will control the WAM arm (Barret Technology Inc., USA) using the Leap Motion sensor will be explained. Later this system will be tested in order to grasp a polo shirt which will be used to train grasping points on the shirt.

1.2 Leap Motion Sensor

Leap Motion sensor is a small computer input device that recognizes hand, fingers and tools such as pens. It contains 2 monochromatic cameras and 3 infrared LEDs. It uses the information from those input and using 'complex math' it outputs the hand/finger 6D position information via USB. The device is able to track individual hand and finger movements independently. Due to patent pending situation algorithms and detailed information is not available.[1]

Figure 1: Leap Motion sensor

1.3 ROS

Robot Operating System (ROS) (Willow Garage, USA) is an open source framework for writing robot software. It is developed in order to make the process of robot programming easier by increasing collaborative works.

The software is developed using *packages* which implements certain functionalities and algorithms such as SLAM, face detection etc. Those packages may be implemented by anyone who wants to contribute.

The executables are called *nodes*. Communication between ROS nodes are done by *topics*. They have certain message structures. Users may use the ones that are already defined or they can create new structures using the basic ones. For example Leap Motion sensor uses Pose message for hand and finger positions. Its structure;

1. geometry_msgs/Pose

```
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w
```

To send the position information to the robot, we also need the reference frame and time information. Therefore a stamp is added to the Pose message. Its structure is as follows:

2. geometry_msgs/PoseStamped

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

The difference between the Pose and PoseStamped object is that PoseStamped has information about the reference frame and the generation time of the message.

2 Leap Motion with ROS Environment

As the robot needs to be controlled with Leap Motion sensor, first it is necessary to provide the data from Leap Motion in ROS environment. Then, a goal position will be created.

2.1 ros_leap Package

The ros_leap package, created by Bosch RTC Team, is used for obtaining Leap Motion data and publishing it as a topic. Main features of this package are;

- Scale the data from milimeters to meters.
- Change the reference frame.
- Publish the Leap() data

The structure of the Leap() data is:

```
[leap_msgs/Leap]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
int64 leap_frame_id
int64 leap_time_stamp
int16 hands_count
int16 fingers_count
int16 tools_count
leap_msgs/Hand[] hands
  int16 id
  float64 sphere:radius
  float64 direction_yaw
  geometry_msgs/Pose pose
  (...)
  int16[] finger_ids
  int16[] tool_ids
leap_msgs/Finger[] fingers
  int16 id
  float64 width
  float64 length
  geometry_msgs/Point velocity
    float64 x
    float64 y
    float64 z
  geometry_msgs/Pose pose
  (...)
leap_msgs/Tool[] tools
  int16 id
  float64 width
  float64 length
  geometry_msgs/Point velocity
    float64 x
    float64 y
    float64 z
  geometry_msgs/Pose pose
  (...)
leap_msgs/Gestures[] gestures
  int64 id
  int64 type
  int64[] pointables
```

From this message *leap_msgs/Hand[] hands/pose* in order to detect hand position, *leap_msgs/Gestures[] gestures* to catch the grasp and lift signal and size of the *leap_msgs/Fingers[]* list to be able to detect fist which will help to remove the hand without moving the robot will be used.

3 leap_wam_controller Package

This is the package that has been developed for controlling the WAM arm. It basically subscribes to the topic that contains the Leap Sensor data and generates a goal position for the WAM robot.

The main flow of this package is as follows:

- Subscribe the hand position information.
- Generate a goal position for the robot according to hand position.
- If grasp signal comes, close the gripper and lift the item.

The detailed flowchart of the leap_wam_controller and its sub-state machines can be seen in Figure 5a 5b and 5c. States will be explained in section 3.2.1

This package publishes a PoseStamped object to the DMPTracker package which is a low-level library for generating trajectories between two positions in joint space.

3.1 Installation

To download leap_wam_controller run:

```
$ git clone https://github.com/yalim/leap-wam-controller.git
```

3.1.1 External Dependencies

To use leap_wam_controller package, you need to install ROS[1], LEAP Motion Software and Developer SDK Bundle[2] and ros_leap package[3].

In order to activate Leap Motion sensor, connect the sensor, open a terminal and run:

```
$ sudo service leapd stop
$ leapd
```

After executing the last command you should see:

```
[13:12:18] [Info] WebSocket server started
[13:12:19] [Info] Processing initialized
[13:12:19] [Info] Leap Motion Controller detected: LP9440...
```

3.1.2 Internal Dependencies

Install iri-ros-pkg[4].

3.2 Use of leap_wam_controller Package

Before controlling the robot with Leap sensor, the robot must be ready. To do this first connect to the robot computer using *ssh* then run the following on the robot:

```
$ roslaunch iri_wam_bringup estirabot.launch
$ roslaunch iri_wam_bringup iri_wam_bringup_gripper_no.launch ROBOT:=estirabot
```

Now open up a terminal and run

```
$ roslaunch iri_wam_bringup iri_wam_bringup_kinect.launch ROBOT:=estirabot
```

In order to move the robot DMP tracker should be running. In a new terminal run:

```
$ roslaunch iri_wam_dmp_tracker test.launch ROBOT:=estirabot
```

Now, polo shirt detection should be started. To do this run while the table is empty:

```
$ roslaunch estirabot_apps pick_up_cloth_skills.launch openni_poincloud_topic:=/
    estirabot/camera/depth_registered/points
```

Place the shirt on the table. To start using leap_wam_controller package, open another terminal and run:

```
$ roslaunch leap_wam_controller leap_wam_controller.launch
```

You can hover your hand over Leap Motion sensor and control the WAM robot. If polo detection is not necessary and you only want to control the robot using the Leap Motion sensor, do not run the *iri_bow_object_detector* package, remove the 'INITIALIZE' state from the Leap Wam Controller State Machine (Figure 5a) and run *leap_wam_controller* package.

[1] `http://wiki.ros.org/hydro/Installation/Ubuntu`
[2] `https://developer.leapmotion.com/downloads`
[3] `https://github.com/bosch-ros-pkg/ros_leap`
[4] https://devel.iri.upc.edu/pub/labrobotica/ros/iri-ros-pkg/rosinstall/

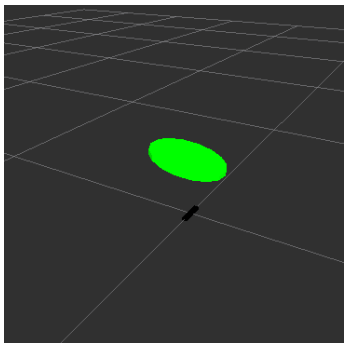
3.2.1 States

leap_wam_controller SM GRASP
This is an SimpleActionState which calls */gripper/tool_close_action* in order to close the gripper.
LIFT
This states publishes a new pose in order to lift the grasped object

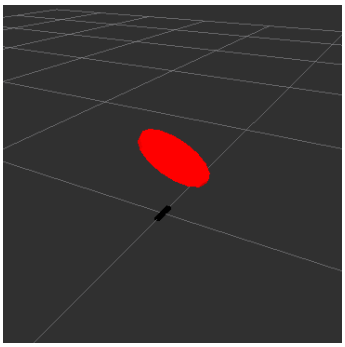
GO_TO_DESIRED_POSE SM INITIALIZE
Subscribes to the current position of the robot and initializes the PositionCreator
CHECK_FOR_COLLISION
As we are manipulating on a table, we have to avoid hitting it. We know that table lays on x-y plane therefore, this state only checks if the pose to be published has a z value lower than a given threshold. If so, it sets the z value to the threshold.
POSE_PUBLISHER
Publishes a new position for the robot on */pose_st* topic.
GRASP_CHECKER
It checks if the grasp signal is received from the hand. The grasp signal is key tapping of one finger(Figure 2). Leap Motion Sensor provides that information via */leap/data* topic.

Figure 2: Key Tap Gesture
(https://developer.leapmotion.com/documentation/python/api/Leap.KeyTapGesture.html)

POSITION_CREATOR SM READING_LEAP
Subscribes to */leap/data* to get the hand information. If there is more than one hand, no hand at all or the hand is in a fist position, it returns *empty* otherwise it passes the hand position to the STD_DEV state.
STD_DEV
It checks the standard deviation of the hand position in order to wait until hand stabilizes when the hand first introduced. Without this state, robot will move as soon as hand is seen by the Leap Motion sensor. It also publishes the hand position to *rviz* interface. The color coding of hand marker indicates if the hand position is stable. If the hand is stable, the color turns green (Figure 3a). It remains red otherwise(Figure 3b). Once hand is stable the user is able to control the robot.
CHECK_IF_MOVED
This state checks whether the robot moved by the user. If moved, it sends it enables the system to reinitialize the starting position when the hand is reintroduced.
STAY_STILL
When there is no hand present, this state re-sends the last position in order to make the robot stay still. It also resets the initial position for the POSITION_DIFFERENCE state.
POSITION_DIFFERENCE
This state generates the new goal position for the robot based on the hand position. It uses the formula:
$new\ position = initial\ position + hand\ difference$
where



(a) Stable hand (Green Disk)



(b) Unstable hand (Red Disk)

Figure 3: Stable and unstable hand visualization

$hand\ difference$ = $hand\ position\ on\ first\ introduction$ – $current\ hand\ position$ It also filters the position with a low pass filter in order to eliminate abrupt changes which causes a failure in DMP tracker.

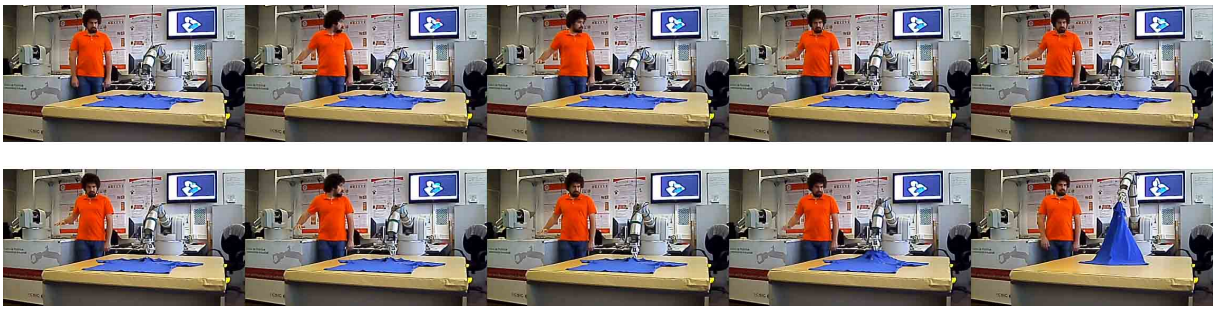
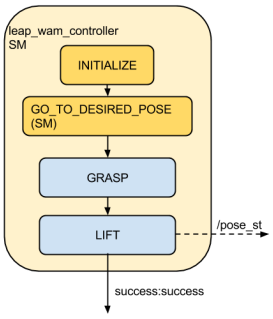
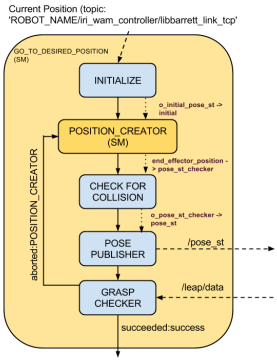


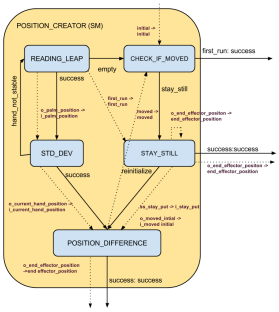
Figure 4: Running of the system



(a) Leap Wam Controller Flowchart



(b) Go To Desired Pose State Machine



(c) Position Creator State Machine Flowchart

Figure 5: Program Flowcharts

4 Results and Conclusions

4.1 Results

Several experiments have been conducted in order to test the performance of the system. For simplicity, manipulation is done only in 4 DoF which are 3 DoF position and rotation around z axis. Due to the

noisy data from the sensor, a low-pass filter was implemented. Moreover, to prevent the robot from crashing, a slower rate is selected for publishing position information.

After the experiments we noticed that the filter and DMP tracker caused a significant delay which made the user incapable of positioning the end effector in a desired position with ease. When the effect of the filter is reduced, delay between hand movements and published pose became insignificant. However a delay of 1.5 seconds due to DMP tracker still present as seen in Figure 6. Furthermore, as the hand moves farther than the sensor, the data obtained became less reliable. For example, the direction of the hand seems to vary even though only the position was changing. We noticed that the best region of operation of Leap Motion is within 40 cm of distance.

As it is mentioned in Chapter 3, a key tap gesture is used to send grasp and lift signal. During the experiments, it took several tries to capture the key tap gesture.

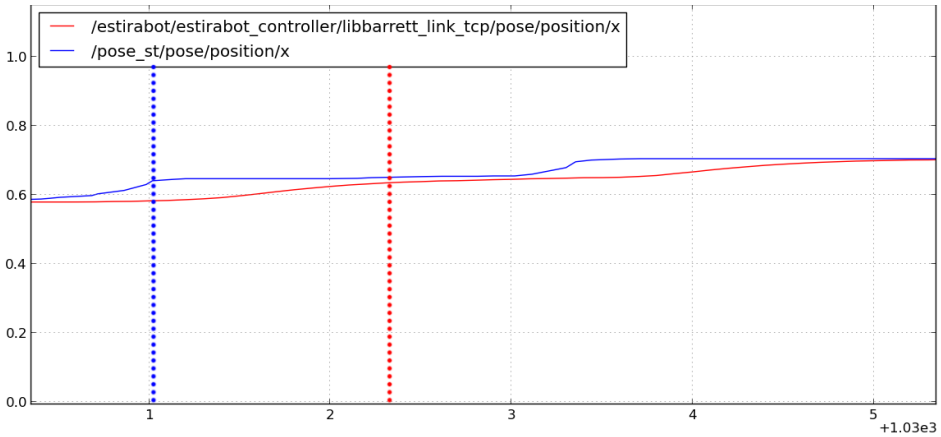


Figure 6: Delay due to DMP tracker. (Blue straight line: Published pose, Red straight line: Robot position, Dotted lines: Input time of published pose (Blue) and Arrival time of the robot (Red)

4.2 Conclusion

In general, we can safely say that Leap Motion sensor is a suitable device for controlling the robot remotely. When the filter is made weaker and the hand is kept in the reliable region, the robot responded well to all signals. However, sending the grasping signal was still problematic. Using just hand movements to control the robot feels very natural and it enhances the user experience.

References

[1] The unofficial leap faq, February 2014.

IRI reports

This report is in the series of IRI technical reports.
All IRI technical reports are available for download at the IRI website
`http://www.iri.upc.edu`.