# reflex-gripper-ros-pkg Documentation

## Contents

# 1 Reading This Document

The purpose of this document is to detail the use of the Reflex SF gripper with the Reflex Gripper Ros Packages. The packages are all catkin packages and are included as a single repository located at `http://github.com/rpiRobotics/reflex-gripper-ros-pkg`.

This document assumes some prior experience using ROS and Robot Raconteur for the details of the code themselves, however, very little knowledge of these topics is needed to use the Reflex Gripper Interface.

The package has been tested using ROS Indigo and Ubuntu 14.04.

# 2 Reflex SF

The Reflex SF is a 3 fingered gripping hand made by Right Hand Robotics. Inside the housing of the Reflex SF are 4 dynamixel servos that drive the three fingers and preshape. The fingers are named and referred to in the code and documentation as follows:
Using your right hand as a model for the Reflex SF extend your index finger, middle finger, and thumb. Your index finger corresponds to '*f1*', your middle to '*f2*', and your thumb to '*f3*'. The '*preshape*' corresponds to a scissoring action between your index and middle fingers.

There are no joint encoders or tactile sensors on the Reflex SF. However, the code included in the `reflex-gripper-ros-pkg` includes everything necessary to communicate with the dynamixel servos which can detect and communicate their current angle in radians, their velocity, and their loads (torques).

For more information on the Reflex SF check out Right Hand Robotics website at `http://www.righthandrobotics.com/main:reflex`.

# 3 Installation

Installing the Reflex Gripper ROS Packages is as simple as cloning the repositiory to the `src` directory of your ROS or catkin workspace (I will refer to this in the documentation as ∼`/ros_ws`)

```
$ cd ∼/ros_ws/src
$ git clone https://github.com/rpiRobotics/reflex-gripper-ros-pkg
```

You should then try to build and install the packages to your ROS workspace using `catkin_make`

```
$ cd ∼/ros_ws
$ catkin_make install
```

If you see any build or dependency issues while building note that the packages do depend on the `ros-indigo-dynamixel-controllers` package ...

```
$ sudo apt-get install ros-indigo-dynamixel-controllers
```

... as well as Robot Raconteur python bindings `https://robotraconteur.com/download/`.

Please ensure that both of these dependencies are correctly installed before continuing. Once your ros workspace is built you may need to source your setup.bash file again.

```
$ cd ~/ros_ws
$ source devel/setup.bash
```

Your package should now be ready to use and scripts can be called using the typical ros commands (`roslaunch, rosrun, ...`)

# 4 Project Packages

The `reflex-gripper-ros-pkg` repository contains several different packages that are required to use the Reflex Gripper Interface.

- `reflex_gripper`
- `reflex_gripper_msgs`
- `reflex_gripper_rr_bridge`

The `reflex_gripper` package contains all of the source code for the Reflex SF ROS Interface. The `reflex_gripper_msgs` package includes the ROS messages and services definitions used by the packages. The `reflex_gripper_rr_bridge` package is a Robot Raconteur Service implementation for the Reflex SF hand that can be used as a middle-man between the ROS interface and a client written in any programming language supported by Robot Raconteur.

More information about each of these packages is detailed in the following sections.

## 5 `reflex_gripper`

The `reflex_gripper` package contains all of the source code to launch the Reflex SF Gripper's ROS interface. The package is structured like many typical catkin packages.

```
/reflex_gripper
├── launch
├── scripts
├── src
│   └── reflex_gripper
└── yaml
```

I will go into more detail for each directory and the code contained within in the following subsections.

### 5.1 `/reflex_gripper/launch`

The `launch` directory contains launch scripts to be used with the `roslaunch` command.

The launch scripts that should be run include `motors.launch` and `reflex_gripper.launch`. The `params.launch` file should not be called directly from

the command line. It loads all of the necessary rosparams for a gripper hand and is called by the other launch scripts. To facilitate launching multiple hands from the same computer, each `.launch` file takes several optional arguments. These arguments are `Name` and `Port`, to specify the name and USB port respectively that the hand should be launched with. The default `Name` is 'hand' located on `Port` `/dev/ttyUSB0`.

#### 5.1.1 `motors.launch`

`motors.launch` will launch only the dynamixel controllers necessary for each finger motor. There are not many reasons to launch only the finger ROS nodes but the option is given to the user.

`motors.launch` is run using a the following command:

```
$ roslaunch reflex_gripper motors.launch Name:=<NAME> Port:=<PORT>
```

#### 5.1.2 `reflex_gripper.launch`

`reflex_gripper.launch` will launch both the dynamixel controllers (it calls `motors.launch`) as well as a node for the entire hand that actually implements the ROS interface. Note that only one hand can be launched per launch file. If the user wishes to launch multiple hands at once the `launcher.py` script should be used.

`reflex_gripper.launch` is run using the following command:

```
$ roslaunch reflex_gripper reflex_gripper.launch Name:=<NAME>
    Port:=<PORT>
```

### 5.2 `/reflex_gripper/scripts`

The `scripts` directory contains several useful scripts to interact with the hand including two calibration scripts and a launcher script that allows the user to interactively launch the ROS interfaces with more control and features than using the `.launch` files. These scripts are run using the `rosrun` command.

#### 5.2.1 `calibrate.py`

The `calibrate.py` script can be called to calibrate a hand either before starting the hand's ROS node or while the ROS node is already running. The script prompts the user to move the hand to its zero position one finger at a time. Once the calibration is complete it saves out these zero points. If a hand is started with the same name as the one used during calibration the zero points will automatically be set to these saved points so that hand may not need to be recalibrated every time.

#### 5.2.2 `calibrate_interactive.py`

The `calibrate_interactive.py` script serves the same function as the `calibrate.py` script. It differs in that it creates a terminal interface for the user to select which finger to move and uses the arrow keys to move the fingers rather than keying in a command.

### 5.2.3 `interactive_controller.py`

The `interactive_controller.py` script uses the same interface as `calibrate_interactive.py` to control the fingers individually but it does not save out any calibration file once it exits. This script is meant more as a testing script to check that each motor is working correctly. To run this script only the finger nodes should be launched as this script launches its own hand node.

### 5.2.4 `launcher.py`

The `launcher.py` script is a very useful script that can be used to launch fingers, hands, or Robot Raconteur Nodes for any number of hands all from a single command line interface. The command line help message is shown below:

```
$ rosrun reflex_gripper launcher.py -h
usage: launcher.py [-h] [--port PORT [PORT ...]]
                   [--bkgOnly | --RRService [RRSERVICE]]
                   [--calibrate | --calibrateI | --nocalibrate]
                   name [name ...]

Launch the ROS components of a ReflexSF Gripper Hand

optional arguments:
  -h, --help          show this help message and exit

Hand Arguments:
  Arguments specific to launching a hand

  name                A list of names for the grippers.
  --port PORT [PORT ...]
                      A list of the USB ports the hands are attached to.
                      Specified in the same order as the names.
                      (defaults to '/dev/ttyUSB<#>' starting at 0)

Script Options:
  --bkgOnly           Set up all the required background nodes but not
                      the main gripper node
  --RRService [RRSERVICE]
                      Start the main gripper node as a Robot Raconteur
                      Service.The optional argument is the RR Port
                      number.
                      Note: you must have the reflex_gripper_rr_bridge
                      package installed.
  --calibrate         Run the default calibration script as part of the
                      launcher script
  --calibrateI        Calibrate the hand as part of the setup script
                      using the interactive controller.
  --nocalibrate       Do not calibrate as part of the launcher script.
                      If a gripper with the given name has not been
                      started before the calibration script will be
                      run by default.
```

As you can see there are many different options the script can be passed to specify what you would like to launch.

The only required field is the `name` of the hand node to launch. You can optionally specify more than one name to launch several hand nodes at once.

The `--port` option allows you to specify a list of the USB ports associated with each respective hand. If this option is not given it is assumed that each hand named is associated sequentially with the `dev/ttyUSB*` beginning with `ttyUSB0`.

Every other option is an option for what you would like the script to do. For example, `--bkOnly` and `--RRService` are used to start only the background finger nodes or launch a Robot Raconteur Service respectively. If neither of these options are present the default action is launching just a ROS node for each hand.

For the `--RRService` option specifically a desired port number can be given after the flag. For example. `--RRService 2223` will start the Robot Raconteur Node at `tcp://localhost:2223/...`. If no port number is given one will be auto-generated and displayed on the screen once the node starts.

The remaining options all deal with whether you would like to run a calibration script as part of the launch. There is the option between the default calibration script (`--calibrate`), the interactive calibration script(`--calibrateI`), and no calibration script(`--nocalibrate`). Note that if any of the hand names specified do not have a calibrated zero points file the `--nocalibrate` option will be ignored and a calibration script must be run. If no calibration option is given at the command line the script will ask you after launching the specified nodes if you would like to calibrate the hand.

## 5.3 `/reflex_gripper/src`

The `src` directory contains code for a tf telemetry broadcaster as well as the `reflex_gripper` python module. I will not describe the code in detail but I will describe the contents of each file.

*Note:* The code that was taken from Right Hand Robotics implemented the motors and hands in a subclass / superclass manner in order to support different types of hands for their different products. We are only concerned with the Reflex SF hand, hence we only included those subclass definitions but this structure of the classes was kept just for compatibility if anyone later on wishes to add a different type of hand to the `reflex-gripper-ros-pkg`.

### 5.3.1 `tf_broadcaster.cpp`

The `tf_broadcaster.cpp` file contains the source code for a telelmetry broadcasting node.

### 5.3.2 `reflex_gripper/curses_interface.py`

The `curses_interface.py` file contains the source code for the interactive terminal used by the `calibrate_interactive.py` and the `interactive_controller.py` scripts. In order to manipulate the terminal it utilizes a program (and python package) called 'curses', hence the name `curses_interface.py`.

### 5.3.3 reflex_gripper/motor.py

The motor.py file contains the default class definition for a dynamixel motor (Motor) and everything that its subclasses must implement.

### 5.3.4 reflex_gripper/reflex_sf_motor.py

The reflex_sf_motor.py file contains the ReflexSFMotor class definition. This class implements the Motor class specifically for the Reflex SF Hand.

### 5.3.5 reflex_gripper/reflex_hand.py

The reflex_hand.py file contains the default class definition for a Right Hand Robotics hand (ReflexHand).

### 5.3.6 reflex_gripper/reflex_sf_hand.py

The reflex_sf_hand.py file contains the ReflexSFHand class definition which implements the ReflexHand class specifically for a Reflex SF Hand. Each ReflexSFHand class internally creates and stores four ReflexSFMotors, one for each finger.

## 5.4 /reflex_gripper/yaml

The yaml directory contains several yaml files that save important information about the Reflex SF hands in yaml format. These include files created when calibrating the hand, reflex_NAME_zero_points.yaml, as well as the yaml that is used to define the motors in relationship to the hand, reflex_sf.yaml, and the telemetry information, tf_geometry.yaml.

# 6 reflex_gripper_msgs

The reflex_gripper_msgs directory contains the definitions for the ROS Messages and Services used by the reflex-gripper-ros-pkg packages.

# 7 reflex_gripper_rr_bridge

The reflex_gripper_rr_bridge is a catkin package that defines a Robot Raconteur service for the Reflex SF grippers. It exposes nearly all of the functionality of the ROS nodes so that a client program can be written to interact with the grippers using any programming language supported by Robot Raconteur. All of the source code for this package is contained in one file, src/reflex_gripper_rr_bridge/gripper_host.py.

The Robot Raconteur Service can be started either from the launcher.py script along with the required ROS nodes, or by directly running the gripper_host.py script.

```
$ rosrun reflex_gripper_rr_bridge gripper_host.py [NAME] --port [PORT]
```

The NAME field must use the same name as the hand for an already running ROS node. The --port option allows the user to specify the TCP port on their

local machine the Robot Raconteur service should be started on. For example. `--port 2223` will start the Robot Raconteur Node at `tcp://localhost:2223/...`. If no port number is given one will be auto-generated and displayed on the screen once the node starts.

For reference the Robot Raconteur Service definition is reproduced below:

```
#Service to provide simple interface to the Right Hand Robotics Gripper
    Hands
service ReflexGripper_Interface

option version 0.4

object Gripper

property double[] joint_positions
property double[] raw_angles
property double[] joint_velocities
property double[] joint_torques

function void setControlMode(uint8 mode)
function void setJointCommand(double[] command)
function void setPositionModeSpeed(double speed)
function void calibrate()

#Functions for position mode presets
function void setGripShape(string shape)
function void closeGrip()
function void openGrip()
function void tightenGrip()
function void loosenGrip()
function void tightenFinger(string finger)
function void loosenFinger(string finger)
function void zeroHand()

end object
```

The following section will describe in a little more detail the available properties and functions.

## 7.1 Properties

The following properties are available from the Robot Raconteur Service.

- `joint_positions`
- `raw_angles`
- `joint_velocities`
- `joint_torques`

### 7.1.1 `joint_positions`

The `joint_positions` property gives the joint angles, in radians, of each finger as a stacked vector. These angles are relative to the fingers zero position.

#### 7.1.2 `raw_angles`

The `raw_angles` property gives the raw angles, in radians, of each dynamixel motor as a stacked vector. These angles are the raw angles the dynamixel motor believes it is currently set at.

#### 7.1.3 `joint_velocities`

The `joint_velocities` property gives the current velocity of each joint, in radians per second, as determined by the dynamixel motor. Note that these are not the commanded velocities for the joint or the preset Position Mode Speed, they are the actual velocities of each joint as determined by the dynamixel motor.

#### 7.1.4 `joint_torques`

The `joint_torques` property gives the current loads on each joint. The values are actually unitless due to the way Dynamixel motors measure torque. For more information on force feedback visit `http://support.robotis.com/en/product/dynamixel/mx_series/mx-28.htm#Actuator_Address_28`.

### 7.2 Functions

The following functions are available from the Robot Raconteur Service.

- `setControlMode`
- `setJointCommand`
- `setPositionModeSpeed`
- `calibrate`

#### 7.2.1 `setControlMode`

The `setControlMode` function takes a single `uint8` to specify the desired control mode of the hand. This mode determines how the commands sent to the hand will be interpreted and carried out. The modes are as follows:

- 0 - default (stacked vector of position and velocity)
- 1 - position
- 2 - velocity
- 3 - torque

Commands while in the default mode must be given as a stacked vector of positions and velocities ($[8 \times 1]$ vector). All other modes expect a stacked vector of only the value specified ($[4 \times 1]$ vector).

In the default mode, the user may specify both a position (in radians) for the finger to achieve, as well as the velocity (in radians per second) for the finger to move to that position. The command vector should be specified as $[positions; velocities]$.

As one would expect, in the position mode, the user commands will be interpreted as desired positions (angles in radians) for each finger. The fingers

will move at the speed set by the `setPositionModeSpeed` function (or by default 4.5 radians per second).

In the velocity mode, the user commands will be interpreted as desired velocities for each finger and each finger will move at that given velocity.

In torque mode, the user command will be interpreted as a desired force to apply (see the note under `joint_torques`). The hand will attempt to keep this load on the motor as long as it is not over the motors maximum torque.

#### 7.2.2 `setJointCommand`

The `setJointCommand` function is used to actually send a command to the hand. Depending on the current mode, the function is expecting a vector of either 8 (default mode) or 4 (all other modes) `double`s. The vector should specify the command for the fingers in the following order `[f1,f2,f3,preshape]`. In the case of the default mode that order should be repeated for both positions and velocities.

#### 7.2.3 `setPositionModeSpeed`

The `setPositionModeSpeed` function is used to specify what speed the motors should move at if in position mode. By default the motors will move at 4.5 radians per second.

#### 7.2.4 `calibrate`

The `calibrate` function is used to call the hand's calibration service. The service will be run in the same window as the ROS node for the hand, not the window for the Robot Raconteur Service.

### 7.3 Position Mode Presets

Several functions are also available that are only available when the hand is being used in position mode. The available preset functions are as follows:

- `setGripShape`
- `closeGrip`
- `openGrip`
- `tightenGrip`
- `loosenGrip`
- `tightenFinger`
- `loosenFinger`
- `zeroHand`

Most of the function's purposes should be self explanatory but they will be quickly described in the following sections.

#### 7.3.1 `setGripShape`

`setGripShape` takes a string as an argument. The string defines a preset grasp shape. The grasp shapes available are cylindrical, spherical, and two finger pinch. The acceptable strings to set these shapes are '`cylinder`', '`c`', '`sphere`', '`s`', '`pinch`', and '`p`'. The default grasp shape is cylindrical (since that is the same grasp as when the hand is in its zero pose).

#### 7.3.2 `closeGrip`

`closeGrip` closes the current grasp shape completely. How far the fingers close is dependent on the grasp shape.

#### 7.3.3 `openGrip`

`openGrip` opens the current grasp completely and returns to the starting position for the grip shape.

#### 7.3.4 `tightenGrip`

`tightenGrip` tightens each finger except for the preshape.

#### 7.3.5 `loosenGrip`

`loosenGrip` loosens each finger except for the preshape.

#### 7.3.6 `tightenFinger`

`tightenFinger` tightens only the finger specified by the string argument. The valid finger names that can be specified are '`f1`', '`f2`', '`f3`', '`preshape`', and '`p`' (a shorthand for 'preshape').

#### 7.3.7 `loosenFinger`

`loosenFinger` loosens only the finger specified by the string argument. The valid finger names that can be specified are '`f1`', '`f2`', '`f3`', '`preshape`', and '`p`' (a shorthand for 'preshape').

#### 7.3.8 `zeroHand`

`zeroHands` will return the hand to its zero position (all angles are zero). This is also the same positions for the cylindrical grasp.