Main

Coop

Script that sets up and control communication between work station and UR5, calls for velocity calculations and image acquisition.

Robot Communication

ActivateCoopMode

Input: t – A tcpip socket between the host workstation and the UR5
Output: []

Sends '(5)' to the UR5, and the variable "task" on the URScript is set to this value. This tells the robot to begin receiving velocity commands from the host workstation, thereby activating collaborative control.

DeActivateCoopMode

Input: t – A tcpip socket between the host workstation and the UR5
Output: []

Sends '(4)' to the UR5, and the variable "task" on the URScript is set to this value. This tells the robot to stop receiving velocity commands from the host workstation, thereby disactivating collaborative control. ActivateCoopMode must have been called before this for this function to have any effect.

getSensorData

Input: t - A tcpip socket between the host workstation and the UR5
Output: out – 12x1 vector containing FT150 sensor data and current joint angle information

Receives and parses out Robotiq FT150 sensor data, along with current joint angle information from the UR5. Saves this information into vector out.

sendVel

Inputs: t - A tcpip socket between the host workstation and the UR5
to_send – Vector containing joint velocity commands that have been calculated
Output: []

Takes a vector of joint velocities and turns into string, which is sent to robot as velocity command.

calculateError

Input: t - A tcpip socket between the host workstation and the UR5
Output: actualSpeed – 6x1 vector containing actual joint velocities of UR5

Reads actual joint velocities from UR5 and calculates the difference between the expected values and the actual values (the error) for accuracy analysis.

Robot Control

forwardKinematics

Input: angles – The 6 joint angles of the UR5
Output: gd - SE(3) matrix that defines transformation between robot base and end effector

Calculates the forward kinematics of the robot given the current joint angles. Uses Product of Exponential formulation of forward kinematics.

skew

Input: omega – 3x1 vector to be converted into skew symmetric version
Output: omegaSkew – 3x3 skew symmetric matrix (se(3)) version of omega

Calculates skew symmetric version of omega. Used in calculateJacobian and forwardKinematics function for product of exponential property of kinematics.

If omega = $\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}, \omega_{skew} = \widehat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$

calculateVelocity

Input: out – 12x1 vector acquired from UR5 in getSensorData function
Output: []

Using the force and torque values from the UR5, a gain matrix is used to calculate the desired probe velocity. Gain relationship between force and velocity shows sigmoidal relationship. forwardKinematics is called to determine transformation between robot base and end effector. This is multiplied by previously determined matrix gX to determine transformation between robot base and end of the probe.
Robot end effector position and euler angles are recorded for data analysis.
End effector velocities are translated into joint velocities using the jacobian matrix of the robot (using calculateJacobian function). These velocities are filtered using the oneEuroFilter function and passed into VF to calculate the constrained joint velocities. The final joint velocities is saved as the global variable Velocity which is sent to the robot in sendVel.

oneEuroFilter

Input: input – 6x1 current joint velocity vector calculated in calculateVelocity to be filtered
Output: filteredOutput – 6x1 filtered joint velocity vector that has been filtered by the One Euro Filter

Implementation of oneEuroFilter, a low pass filter with variable cut-off frequency. When the robot is moving slowly, the cutoff frequency is lower to reduce jitter. When the robot is moving faster, the cut off frequency dynamically increases to reduce lag.

LowPassFilter
Class used by oneEuroFilter. Objects of this internally maintain previous value of velocity to be used in future calculation. Has filter function.

alpha
Inputs: rate – current loop rate of the running code
cutoff – Current cutoff frequency for filter
length – Length of input vector
Output: alphaValue – Final alpha value that is used to vary the weight of previous and current velocity values.

Coefficient used by the filter function of the LowPassFilter class to vary how filtered velocities are weighted.

calculateJacobian
Input: angles – 6x1 vector of current UR5 joint angles
Output: jacobian – 6x6 Jacobian matrix that relates the end effector velocities to joint velocities.

Calculates the Jacobian matrix, the matrix of the derivatives of the robot kinematics equations, based on the current joint angles using the power of exponentials formuation. Uses adjoint function to do this.

adjoint
Input: g – SE(3) matrix that defines transformation between joints and/or base and/or end effector
Output: adj – 6x6 adjoint matrix relating two robot joints

Calculates the adjoint matrix. Given g = $\begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix}$, the adjoint matrix is $g_{adj} = \begin{bmatrix} R_{3x3} & (\hat{t} * R)_{3x3} \\ 0_{3x3} & R_{3x3} \end{bmatrix}$ where $\hat{t}$ is the skew symmetric version of t calculated using the skew function.

<u>Data Analysis</u>
plotTraj
Input: tf – 6x1x… matrix containing all the distances between the robot base and end of probe and the euler angles at all collected data poses over the robot's trajectory.

Plots the probe's translation over time in 3d plot and allows us to see straightness of line in 3D space. Also allows us to compare probe rotation to expected rotation.

Virtual fixtures

StayOnLine

Inputs: gd – SE(3) matrix that defines transformation between robot base and probe end
Output: Hline – Matrix that defines the plane normal to the line.
hline – Vector that defines margin of error, line direction, and current error

Define direction of line as unit vector to follow along and a starting point on the line. Read current end effector position and calculate closest point on the line, according to

$$Pcl = L_0 + \frac{(x_p - L_0) \cdot l}{l \cdot l} * l$$

And calculate delta_l to be the difference between current position and Pcl.

Calculate rotation matrix $R = [v1 \quad v2 \quad l]$, where $v1 = \frac{l \times lprime}{\|l \times lprime\|}$ and $v2 = \frac{l \times v1}{\|l \times v1\|}$

$$H_{line} = \begin{bmatrix} [R * [sin\left(\frac{2\pi}{n}\right) & sin\left(\frac{2\pi}{n}\right) & 0]']' & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ R * [sin\left(\frac{2\pi n}{n}\right) & sin\left(\frac{2\pi n}{n}\right) & 0]']' & 0 & 0 & 0 \end{bmatrix}$$

$$h_{line} = \begin{bmatrix} \varepsilon \\ \dots \\ \varepsilon \end{bmatrix} - H_{line} * \begin{bmatrix} delta\_l \\ delta\_r \end{bmatrix}$$

JointVelLimits

Inputs: []
Output: Hjoint, hjoint

Setting maximal joint velocities between -1 and 1 rad/sec

$$Hjoint = \begin{bmatrix} -I_{6x6} \\ I_{6x6} \end{bmatrix} \text{ and } hjoint = \begin{bmatrix} -q_{min} \\ q_{max} \end{bmatrix}$$

VF

Inputs: gd, jac, linearVelocity
Output: []

Uses lsqlin() to optimize solution for joint velocities, according to the form:

$$\min_{\dot{q}} \frac{1}{2} \|J\dot{q} - \dot{x}\|^2{}_2$$
$$s.t.\ H\dot{q} \leq h$$

Where $H = \begin{bmatrix} H_{line} & 0 \\ 0 & H_{joint} \end{bmatrix} \begin{bmatrix} J \\ I_{6x6} \end{bmatrix}$ and $h = \begin{bmatrix} h_{line} \\ h_{joint} \end{bmatrix}$