Balancing Atlas
Wyatt Newman Nov 5, 2013

Getting Atlas to stand stably under user control is challenging. We suspect numerical problems with drcsim 3.1. Nonetheless, it is possible to stabilize Atlas, as evidenced by the "dynamic stand" behavior that is initiated when launching drcsim.

This document describes some tools that should be useful in helping to develop a stable balancer for Atlas.

Launch drcsim:
Bring up the simulator with the "cheats" enabled (so the torso can be pinned, optionally).
VRC_CHEATS_ENABLED=1 roslaunch drcsim_gazebo atlas_v3_sandia_hands.launch

If desired, pin the torso from a separate terminal (after Atlas has begun standing—about 10 seconds after launch). Do:
rostopic pub --once /atlas/mode std_msgs/String '{data: "pinned_with_gravity"}'

This can be un-done later, if desired, by running:
rostopic pub --once /atlas/mode std_msgs/String '{data: "nominal"}'

Visualizing center of mass and centers of pressure: A useful utility is visualization of Atlas's center of mass and centers of pressure for the feet, projected onto the ground. To get this data, first do:
rosrun mass_calcs fc_com_calc
This will start a node that computes and publishes center-of-mass information for Atlas. New topics under /com/ are published, and these may be used for visualization as well as for feedback control.

Next, launch rviz with the following command:
roslaunch com_cop_visualizer com_cop_visualizer.launch

This will bring up rviz, displaying several colored dots that are useful. A green dot shows the computed center of mass (which is in the chest, above the pelvis). Three more dots show the center of pressure under each foot and the average of these. For stable stance, the average center of pressure (or the projection of the center of mass onto the ground plane) should be roughly mid-way between the two ankles and ideally about 4cm in front of the ankles.

Another useful node to run is:
rosrun ladder_climbing com_foot

This node subscribes to the COM topics and transforms the information into a new frame: the "right foot gravity" frame and the "left foot gravity" frame. In these frames, the COM is expressed in an orientation that has the z-axis parallel to gravity, and has the x-axis along the major axis of the chosen foot, projected onto the horizontal plane. (The y axis follows from the cross product). The origins of these frames are the right ankle and the left ankle, respectively. This computation is useful for trying to balance the robot on one foot. From this viewpoint, it is desirable that the COM be approximately 4cm positive along the x axis of the chosen foot and ideally zero with respect to the y axis.

Gravity torques:
A node in the package “mass_calcs” subscribes to the robot state (including joint angles and IMU data for direction of gravity) and computes the joint torques required to oppose gravity. This is performed for the legs as well, assuming that the two feet share supporting the weight and the tipping moments of the COM. This node can be run with:
rosrun mass_calcs stand_torque_calc
It then publishes multiple topics, including: "torque_control/stand_both", which publishes a vector of joint torques that approximately oppose gravity.

An example of how to subscribe to this topic and use it in control is in the package “lowLevelJointServoV2”. As an illustration, if the computed gravity torques are added to the arms (with pelvis pinned) but no other servo control is applied, the arms will seem to “float” as though they were in freefall (zero-g).

Jacobian publisher:
The node launched by:
rosrun jacobian_publisher jacobian_publisher
publishers several useful topics. Use of these Jacobians is also illustrated in the package lowLevelJointServoV2. The Jacobian publisher subscribes to Atlas' state and computes and publishes Jacobians for each limb, from pelvis to extremity (foot or hand).

Inertia publisher:
The node run by:
rosrun mass_calcs H_calc
subscribes to Atlas' state and computes inertial terms. These terms can be used to modulate control gains, since the inertias of the limbs vary dramatically as a function of pose. Examples of how to subscribe to the inertia topics are in lowLevelJointServoV2. At the time of this writing, the inertia computations have not been fully validated. The diagonal terms seem reasonable, and the off-diagonal (coupling) terms look plausible from spot checking. More testing is required. It is expected that at least the diagonal terms of the inertia matrices returned are useful for modulating gains.

LowLevelJointServoV2:
This new package was written to try to reconcile low-level controllers between drcsim 3.1 and the physical Atlas robot. It can be run with:
rosrun lowLevelJointServoV2 lowLevelJointServoV2
This routine accepts commands via the existing command topic "/lowLevelJntCmd", used previously.

This controller has several innovations, but is not yet fully tested nor fully functional. Innovations include:
low_level_jiont-servo_class_v2_main.cpp:

- simply instantiates an object of class LowLevelJointServoClass, which performs all initializations and subscriptions in its constructor.
- Loops at 333Hz to emulate the communications rate of physical Atlas to the “field computer”.
- All computations for servo control are performed by callbacks within “jointServoCallback.”

LowLevelJointServoClass.h:

- contains in-line code for unpacking a matrix message to populate H matrices
- Includes some simple utilities to extract various subsets of joints (e.g., the right arm

joints only) from the full vector of joints, and how to re-insert results from subsystems into the full 28x1 vector.

low_level_joint_servo_class:

- shows how to define and use callback functions that are methods of a class. This is done with all 12 of the callbacks currently used in this node
- shows how to share a single “transform listener” among all methods of the class by making it a class object.
- Shows how to transform some Jacobians (e.g. in jacobianListenerRFootCB). Published Jacobians J_rfoot_wrt_pelvis and J_lfoot_wrt_pelvis are transformed into Jacobians of the pelvis with respect to the foot, e.g. J_linear_pelvis_wrt_rfoot.
- Demonstrates an example of using the class TaskVariableXYZRPY to get arbitrary Jacobians of some named frame with respect to a named reference frame.
- Demonstrates try/catch loop for testing if the transform listener has received a desired set of frames.
- Includes a utility function for populating a skew-symmetric matrix from a vector.

servo_control_methods.cpp:

- jointServoCallback() Is triggered by arrival of atlas_state messages, but slowed down by ros::spinOnce() calls within the main program.
- Relies on calls to “compute_all_servo_efforts()” to perform joint effort computations.
- Optionally uses the drcsim controller (via kp_position, kd_position and position and velocity commands) or pure effort commands, by publishing to atlas/atlas_command.

compute_all_servo_efforts():

- adds computed (published) gravity loads to all torque commands
- computes Kp*q_err + Kd*qdot_err PD (proportional-plus-derivative) control for all joints, which defines a desired correction acceleration for each joint
- adds user-defined desired accelerations to the feedback correction accelerations
- scales all desired accelerations with the corresponding diagonal term of the respective joint's inertia matrix
- makes corrections for effort-sensor bias and Coulomb friction (necessary for physical Atlas, but has no effect for drcsim)
- computes a “skyhook”-based damping contribution for leg efforts with respect to pelvis velocity
- Adds all of these effects
- accounts for actuator saturation levels approximating physical Atlas's limits
- the resulting vector of 28 joint efforts, in e_cmd_, is available for publication to Atlas as a command by the callback function jointServoCallback();
-

Status:

At present, the low-level joint servo class looks promising, but it does not yet stabilize Atlas. One may, for example, pin the torso and suppress all feedback while allowing gravity torques to be exerted. The arms appear to be in free-fall, which apparently validates the gravity model.

A launch script exists: *roslaunch lowLevelJointServoV2 lowLevelJointServoV2.launch*. This launches the H calculator, the jacobian publisher the gravity-load publisher, and the COM transformation node. It does not launch the low-level joint servo, since this node prompts the user for input, and thus this node should be rosrun separately.