EXPERIMENT  
Test Newly coded Gazebo/ROS guarded move that uses Force values to detect collision and terminate motion using Joint 6 of Fanuc LRMate 200id

# Background

Since the explanation of the force/torque sensor use in Gazebo can worked but is haltingly slow, it was thought that an embedded Gazebo plugin :

# Setup:

First, build and install the Gazebo/ROS guarded move plugin with catkin\_build.

Second, add Gazebo/ROS guarded move plugin to Fanuc LRMate robot

<gazebo>

<plugin name="gazebo\_ros\_guarded\_move"

filename="C:\opt\ros\noetic\gztaskboard\pluginswin\gazebo\_ros\_guarded\_move.dll">

<service\_topic>/lrmate/gazebo\_ros\_guarded\_move</service\_topic>

<robotNamespace>lrmate</robotNamespace>

<debug> 1 </debug>

<updateRate> 10 </updateRate>

<jointName>fanuc\_joint\_1, fanuc\_joint\_2, fanuc\_joint\_3, fanuc\_joint\_4, fanuc\_joint\_5, fanuc\_joint\_6 </jointName>

<forceTorqueJoint>fanuc\_joint\_6</forceTorqueJoint>

</plugin>

</gazebo>

Gazebo\_ros\_guarded move SDF parameters include:

* <service\_topic> defines the Service topic name to invoke when calling the plugin
* < robotNamespace> robot namespace
* <debug> turn on/off diagnostic message to the ROS console
* < updateRate> frequency bandwidth of the guarded move updates in times per second.
* <jointName> list of the robot joints to update during guarded move and whose positions will be specified in a joint trajectory to approach the Z limit
* < forceTorqueJoint> is the joint that will be monitored for changing forces with appropriate parameterization to enable force output (gravity, non-static, non-kinematic, ) and the <provide\_feedback> tag enabled for the forceTorqueJoint.

To understand the enabling of force/torque values, (e.g., joint6 fanuc robot since this joint will be used to read force values) see the ExperimentGzFTSensor plugin experiments for further details.

To be admitted, non-zero FT values were reported but no clear relationship to collision with peg array on table.

# SOFTWARE

Inside the peg in hole assembly code, test code was developed to use the guarded move to monitor when the robot grasped peg is touching (colliding with) the peg board array.

To communicate with the Gazebo/ROS guarded move plugin a new Service message was established. It contains the guarded move direction (one dimension for now) and limit force value to stop motion. Since there is no kinematics in the Gazebo model, a joint trajectory using joint names was passed in to establish a trajectory for the guarded move plugin to use.

**GuardedMove.srv**

string move\_name # name of guarded move - unidirectional move for now

geometry\_msgs/Point direction # guarded move in this direction e.g., 0,0,-1 down

geometry\_msgs/Point maxforce # maximum force in given direction. 0 signifies zero.

string[] joint\_names

trajectory\_msgs/JointTrajectoryPoint[] jt\_traj # joints trajectory

geometry\_msgs/Pose[] pose\_traj # poses trajectory

duration duration # optional duration of guarded move application time (seconds)

# if duration < 0, guarded move continuously without end

# if duration = 0, do nothing

# if duration < step size, perform guarded move

# for one step size

---

bool success # return true if set wrench successful

string status\_message # comments if available

The Service message parameters were:

* direction is a vector specifying the direction of the approach, for our case (0,0,-1) or approach in the -Z direction
* maxforce the force threshold to exceed to stop the guarded move.
* joint\_names list of joint names (which should match the sdf) for getting the correct joint from the Gazebo model (joint motion generally not zero based).
* jt\_traj – list of joint positions from current robot position to a robot position that exceeds the guarded threshold. It is expected that a guarded move will stop when the force threshold is violated.
* pose\_traj - a list of pose positions that each guarded move will attain after an update cycle.. Not used by guarded move plugin.
* duration – time limit for the guarded move

The test code was a bit clunky as it approached the peg hole array board (with an offset I x so it would hit the board and not a hole). Previously dead reckoning insertion of the round pet into a round hole was coded and worked. Upon approach to the skewed offset hole position in X, a slow decremented approach in the -Z direction was undertaken until the FT force in the Z direction was significant (0.05 is the current guess.

A naming convention is used to describe the parameters. If it starts with a lower case "w" is indicates a world coordinate space pose. If it starts with a lower case "r" is indicates a robot coordinate space pose. it starts with a lower case "wv" is indicates a world coordinate space Vector.

First the location of pegarray hole table top is determined as a combination of the robot wrist bend (-90 pitch).

tf::Pose rTabletop = robot.\_arm->toRobotCoord(tf::Pose(qBend, wvTopHolePos));

An offset below table is then given to insure colliding with the table top.

rTabletop = robot.\_arm->retract(rTophole1, tf::Vector3(0.0, 0.0, -0.2));

std::cout << "rTabletop Robot Coord = " << conversion::dumpPoseSimple(rTabletop) << "\n";

Then a guarded move service call to move the Gazebo robot down to discernable +z force to a given location and which includes x kludge factor offset from the hole is setup.

First we create a guarded move plugin service which uses the GuardedMove.srv message generated header : #include <gazebo\_plugins2/GuardedMove.h>

gazebo\_plugins2::GuardedMove gm;

Next, we need to fill in the request parameters. First the direction is down which is -Z. The maximum force (guess) is 0.05.

gm.request.direction = geostore(0.0, 0.0, -1.0 );

gm.request.maxforce = geostore(0.0, 0.0, 0.05 );

Next we need to pre-calculate a joint move from the current robot position to the rTabletop position with a .1 of the velocity max (0..1).

moveit\_msgs::RobotTrajectory poses = robot.\_arm->calcMoveTo(rTabletop, .1);

Using the joint trajectory generated by the calcMoveTo to the rTabletop this needs to be converted to ROS geometry message format and then loaded into the guarded move gm request:

std::vector<geometry\_msgs::Pose> pose\_traj;

std::vector<trajectory\_msgs::JointTrajectoryPoint> jt\_traj;

trajconvert(poses, pose\_traj, jt\_traj);

gm.request.jt\_traj = jt\_traj;

gm.request.pose\_traj = pose\_traj;

Note, the joint names are supplied as there are many joints in the robot model that are not part of the robot motion control (e.g., base, gripper, etc.) and simplified the code for the guarded move plugin assigning of joint values.

gm.request.joint\_names = robot.\_arm->jointNames;

Then the service call is invoked. Since its only an experiment it only sends this guarded move and quits the application:

robot.\_arm->\_gzRosGuardedMoveSrvClient.call(gm);;

Yet again, conversion between robot mathematical linear algebra and motion trajectory control representation is required. ROS is tf and geometry and moveit based. Some of ROS is eigen based. Gazebo is now ignition based. NIST CRPI has its own math model. In addition, there are geometry message and moveit pose and joint trajectory message format. So there are a lot of conversion routines, which were extended with trajconvert to convert from a vector of moveit RobotTrajectory to a vector combination of geometry\_msgs::Pose and trajectory\_msgs::JointTrajectoryPoint. One of the issues in the conversion from joint values to pose is the final coordinate system of the pose – world or robot. In the end, the pose was only used in the guarded move plugin as a diagnostic to understand if the motion was moving in the correct manner so the robot coordinate system is used, although gazebo model poses are in the world coordinate system.

void trajconvert(moveit\_msgs::RobotTrajectory intraj,

std::vector<geometry\_msgs::Pose> &pose\_traj,

std::vector<trajectory\_msgs::JointTrajectoryPoint> &jt\_traj)

{

for (size\_t i = 0; i < intraj.joint\_trajectory.points.size(); i++)

{

// assumes robot coordinate system

trajectory\_msgs::JointTrajectoryPoint pt = intraj.joint\_trajectory.points[i];

jt\_traj.push\_back(pt);

// FK into robot coordinate system pose

geometry\_msgs::Pose pose = conversion::Convert<tf::Pose, geometry\_msgs::Pose>(

robot.\_arm->FK(pt.positions));

pose\_traj.push\_back(pose);

}

}

Simple function to simplify initializing geometry\_msgs::Point data structure:

geometry\_msgs::Point geostore(double a,double b, double c)

{

geometry\_msgs::Point p1;

p1.x = a;

p1.y = b;

p1.z = c;

return p1;

}

# RESULTS

The guarded move plugin established that a guarded move plugin could receive the parameters, update the robot pose using passed in joint trajectory motion, but did not prove that a meaningful force could be read in the plugin. A simple test was attempted to see why the force value did not correspond to the collision into the pegarray. This force test allowed the plugin to "sleep" and the hope was it would allow the actual force from joint 6 to percolate through the physics engine. Unfortunately, this was not the case, and requires a further investigation into the physics model of the joint force calculation. This was out of scope. A summary of the experimental observations follows.

With joint trajectory for the guarded motion generated at a .1 velocity scale, over a hundred intermediate motion points were tested against the force threshold value. One benefit is that guarded move plugin is noticeably faster in updating Gazebo motion. This was an original intent to achieve a more realistic simulation of the actual robot behavior, which itself is also slow as the robot cannot immediately stop so approaching a hard physical object is done carefully. On the nor can crash into the table without consequences.

(Note each transition to a new point in the guarded move plugin was halted by a synchronous getch from the ROS console which "slowed" down the motion to observe when contact happened and what poses and forces were at that point in time. The code (which was limited by a service request enable and a rather slow bandwidth update) was:

// Synchronous get character from ROS console - will pause here waiting

int c = \_getch(); // call your non-blocking input function