NIST ARIAC problem assignment: box inspector

This assignment will help get you introduced to the NIST "Agile Robotics for Industrial Automation Competion" (ARIAC) environment. Details and tutorials can be found at the ARIAC wiki here: https://bitbucket.org/osrf/ariac/wiki/2018/Home.md. The objective is to control an automated system with a robot on a track, a variety of cameras and other sensors, as set of bins containing an inventory of parts, and a conveyor with a cardboard-box dispenser. The system should receive and act on requests for "orders." An Order may require multiple boxes to fulfill, where each box is referred to as a "Shipment". A Shipment consists of a list (a C++ vector) of "Products", where each product has a name (a C-string) and a specification of where the product should be placed in the box. It is the job of the system to:

- Control the conveyor to move a shipment box to a reachable location (preferably, halting the conveyor such that the box is within the field of view of an inspection camera)
- check inventory for existence of a part (a Product or a Model) in some bin
- control the robot to move to acquire the part from the identified location
- control the robot to place the held part into the waiting shipment box at the desired coordinates with respect to the box.
- Respond to error sensors, including "dropped part" and "quality sensor pass/no-pass".

As part of this operation, it is necessary to inspect each shipment (box) to make sure that, ultimately, it contains all of the desired parts (and no surplus parts) with each part located as desired within the box to a tolerance of 3cm position and 0.1rad orientation.

Boxes should also be inspected during order fulfillment to check on progress, to see if any parts were accidentally dropped in the box, and to make sure all parts are placed accurately (within tolerances).

A "box inspector" library is to be composed. This has already been roughed out, but there are methods (functions) that need to be filled in. The start of this package is called "box_inspector". It depends on two other packages as well: "osrf_gear" (which defines the various message types used with ARIAC) and "xform_utils" (which is part of learning_ros, within Part2). The box_inspector and osrf_gear packages are contained within the zip file provided. You can expand this file under: ros_ws/src, then compile it.

There are several nodes within box_inspector. To run these, do the following:

With a roscore running: rosrun box_inspector box_inspector_example_main

This starts a node that uses the box_inspector library. This node will wait for an order to come in, then it will wait for a camera image to come in, then it will analyze the sensor data relative to the shipment request.

With box_inspector_example_main running, provide it with an order. There are 4 pre-defined orders you can send it, including: order_sender_ideal, order_sender_4parts, order_sender_6parts, and order_sender_errs. Start by sending the "ideal" order, which should agree precisely with what the (fake) camera claims to see. Run this with:

rosrun box_inspector order_sender_ideal

This node will publish the order, and the main box-inspector node will acknowledge receipt and print the order contents to the screen.

Note that the order specifies desired part names and part locations expressed **with respect to the shipment-box coordinate frame**. It is thus necessary to know where the shipment box is located in space. This information will be available from the inspection camera.

The box-inspector node will attempt to call: boxInspector.get_box_pose_wrt_world(). This function call will block, waiting for (logical) camera data to come in.

To emulate a camera seeing parts in a box, run the node:

rosrun box_inspector fake_logical_camera

This will publish abstracted camera data that exactly emulates the ARIAC simulator, using the same topic name and the same message type.

The box-inspector node will acknowledge receipt of a (fake) camera transmission, and it will print the received data to the screen. The member function get_box_pose_wrt_world() source code shows how to interpret camera data. The camera message will include a "model" with the name "shipping_box." The get_box_pose_wrt_world() function parses the camera message to find the pose of the shipping box. This pose is expressed with respect to the camera's frame. Conveniently, though, the camera's frame with respect to the world is also contained within the camera's message. Combining these poses (box with respect to the camera, and camera with respect to the world), we can compute the pose of the box with respect to world coordinates. This is accomplished with the help of the function "compute_stPose(cam_pose, model_pose)", which takes the two poses and combines them into a PoseStamped message for the model with respect to the world. Note that a stamped pose contains a string declaring its reference frame_id, so there is no ambiguity about how to interpret the coordinates in this object. The means to transform coordinates into world coordinates is similar to the means needed to interpret camera images to compute part locations with respect to the box (to check if they are placed within tolerance specifications).

Next, the box-inspector node invokes the method compute_shipment_poses_wrt_world(), which requires inputs of the shipment specification and the box pose with respect to the world. The need here is to compute where the requested parts belong, expressed in world coordinates, so the robot can be told where to place the parts. **This function needs to be written.**

Next, the box-inspector node calls member function model_poses_wrt_box(). **This function needs to be completed**. The objective of this function is to summarize all of the parts observed by the camera, and express their poses with respect to the box frame. This is the same format as a "shipment", so the result is to be returned in an object "shipment_status". This should be a useful step in performing the box-inspection task.

Finally, the most complex function is called: update_inspection(). **This function needs to be written**. The intent of this function is to interpret the current shipment request in terms of the most recent camera observation, and to classify observed models by including them in one of the following categories:

- a list (C++ vector) of parts that belong in the shipment and which are precisely placed within tolerances of their requested poses. These parts are complete and should be left alone.
- a separate list of parts that are in the box, belong in the order, but are misplaced. These will need to be repositioned by the robot.
- a separate list of parts that are missing from the order (they are specified in the requested order, but they are not currently in the box). These will need to be found in inventory, acquired by the robot, and placed appropriately in the box.
- a separate list of parts that are in the box, but do not belong in the box. Although this would be an unusual situation, it would require that the robot remove these parts from the box.

The nodes in the box_inspector package do compile and run without errors. However, the incomplete functions fail to perform the required logical computations.

In your solution, you may add more member functions, if desired. You should not need to define any new message types, nor any new subscribers. By conforming to the architecture of the starter code, your resulting box-inspector library should be verbatim compatible with the ARIAC simulator.

With the various order publishers, you should be able to conclude that the "order_sender_ideal" results in the inspector deducing that all requested parts were observed in the box by the (fake) camera, and that all of the parts are within tolerance of their requested positions. The other 3 order sender nodes will result in various errors, including missing parts, extraneous parts, and mislocated parts, in addition to parts that are properly located.

Submit your code (pointer on github). Write a brief report describing your solution approach. Present data validating your solution.