Kevin Bradner
Will Huang
ARIAC Time Estimator
12/18/16

Description of produced functionality:

Our time estimator takes in two integer arrays of length 3 and returns a ros::Duration value. The two integers of length 3 correspond to a start and and end pose. We believe that basically all useful poses that the robot could want to assume are pretty well described by two integers from a limited selection of values, so the first two elements of each of the two arrays provided as an argument to our service do most of the work. The third element in each of the supplied integer arrays will simply be a flag, taking the value 0 or 1. 0 is the default option, corresponding to a normal motion. 1 signifies that a pick or place operation is happening for this motion.

The first position of each array corresponds to a position on the track. The possible integer values for the first position of each array have the following meanings:

0: the left AGV
1: the leftmost bin
2: the second bin from the left
3: the second bin from the right
4: the rightmost bin
5: the right AGV

The second position of each array corresponds to joint angles for the arms. The possible integer values for the second position of each array have the following meanings:

0: The bin farther away from the track
1: The bin nearer to the track
2: A “home position” over the track
3: The conveyor belt
4: Arm over an AGV (sort of perpendicular to the other poses)

The third element is, as mentioned, a pick/place flag. When either the first or second argument to the function contains a 1 in the third position, a check is made to see if the second element of the first array holds a value of 3. Assuming it does not, then a time is returned which corresponds to the time required to perform a pick or place in most locations of interest. Otherwise, a pick or place on the conveyor belt is happening. In this case, we assume the worst case scenario and return a time corresponding to the arm reaching both down and as far laterally as it would ever have to do for a sensible strategy.

Assuming that neither array contains a 1 in the third position, then the move involves no pick or place. In this case, the move is “normal,” and our function returns an estimate of the time required to move between the two poses. This estimate is based on 13 timings in gazebo. 9 of these timings are motions along the track position, and the other 4 are motions of the arm itself. We believe that, for practical purposes, all possible motions between desirable poses will take one of these 13 amounts of time, ignoring picking and placing for now. The only caveat to this statement is that these 13 movements are “primitive motions” which done in sequence to execute an entire plan. However, it is fairly straightforward to combine these in sequence to estimate the time taken for an entire plan.

In case you are wondering about the potential utility of moving anywhere along the track to access a certain part of the conveyor belt, note that this part of the estimate is worked into the pick and place component.

In this way, our estimator can process a primitive action from a plan and return the amount of time required to execute it. To analyze an entire plan, you only need to supply the corresponding primitive actions one at a time.

Code Written and Methodology:

Our initial approach for this project was to use Shipei’s pose_tunner.cpp node to learn how to control the UR10 in Gazebo. We actually discovered that this code had numerous inconvenient properties regarding how the motion was carried out. Will spent a considerable amount of time learning to understand Shipei’s code, and then he modified it substantially to produce the behavior that you would expect. Will also spent lots of time tuning the joint angles that correspond to all sorts of desirable poses and extended this node to support moving to those poses. Kevin made lots of changes to this code to ensure that the UR10 moved as fast as possible (Note that it is probably too aggressive in some cases. However, we couldn’t see a way to modify this without a tremendous amount of trial and error effort). Kevin also spent a ton of time making modifications to the code so that it would time the execution of a motion and provide that value back to us. This was not trivial, as it required interacting with additional ROS nodes to get the information we wanted about the state of the robot.

With this much functionality, we devised the plan by which we could accurately estimate the time required for any motion by taking just a few measurements, and then we ran the above code to take these measurements. After that, Kevin wrote code to return the time required for any primitive motion by breaking it down into cases which were equivalent to one of the measurements we had taken.

Will then wrote a basic service and a test client to make all of this functionality usable by a planner node.

The code we wrote can be run with just Shipei’s ariac repo, and we have made a copy of the parts we modified which is available at:
https://github.com/HutEight/ariac_time_estimator.git (to clone)
https://github.com/HutEight/ariac_time_estimator (to view)

The source files which represent a substantial amount of our contribution include (all under cwru_ariac):

/srv/time_estimator_service_msg.srv
/src/time_estimator_client)kmb172_sxh759.cpp
/src/time_estimator_server.cpp
/src/set_joint_angles_velocity.cpp

A video demo of a couple of the ariac motions we produced can be viewed at:
https://www.youtube.com/watch?v=GntiV6LIaYs&feature=youtu.be