UNIVERSITÀ DEGLI STUDI
DI GENOVA

Flexible Automation
Technical report

Lorenzo Causa, Robotics eng.

December 7, 2021

1 Plant Layout Design and Simulation

1.1 Control of input conveyor

The motion of the input conveyor is controlled with its child script, which simply stops the conveyor when the ray proximity sensor detects something. The proximity sensor in the simulation is child of the input conveyor.

1.2 Spawn of the parts

The objects are added to the scene thanks to the non-threaded child script in the dummy *spawnParts*. This script randomly copies and pastes the models that have been previously added. The original models of the parts are on a table, so they can be easily edited (copies will follow accordingly). To be as realistic as possible without making the simulation too heavy, the dynamic twins of the objects have been extrapolated from the original shapes with convex decompositions. Using pure coppelia shapes would have been too approximate while using directly the shapes of the objects would have slowed down too much the simulation. The weights of the parts have been properly set with realistic values.

2 Robot Simulation

2.1 Manipulator info and schematics

To maximize the flexibility/extensibility of the production cell it has been decided to build a $6DOF$ (Degrees Of Freedom) robot, even though a $3DOF$- RPP robot

would probably have been sufficient for the problem. A $6DOF$ manipulator is much more reusable. In the future, the cell may be updated or deleted, the arm proposed in this solution could be reused for almost all kinds of activities, instead, a $3DOF$ manipulator is a lot more specialized, it would risk becoming useless. It was therefore decided to spend some more for a more dexterous system, that has a minimal risk of becoming obsolete/unusable. In order to keep the simulation light, the manipulator has been made with pure shapes of coppeliaSim, very simple models of the pieces have been made with CREO to be used as mesh.
Below you can see the schematics of the arm and the table with the relative DH parameters.

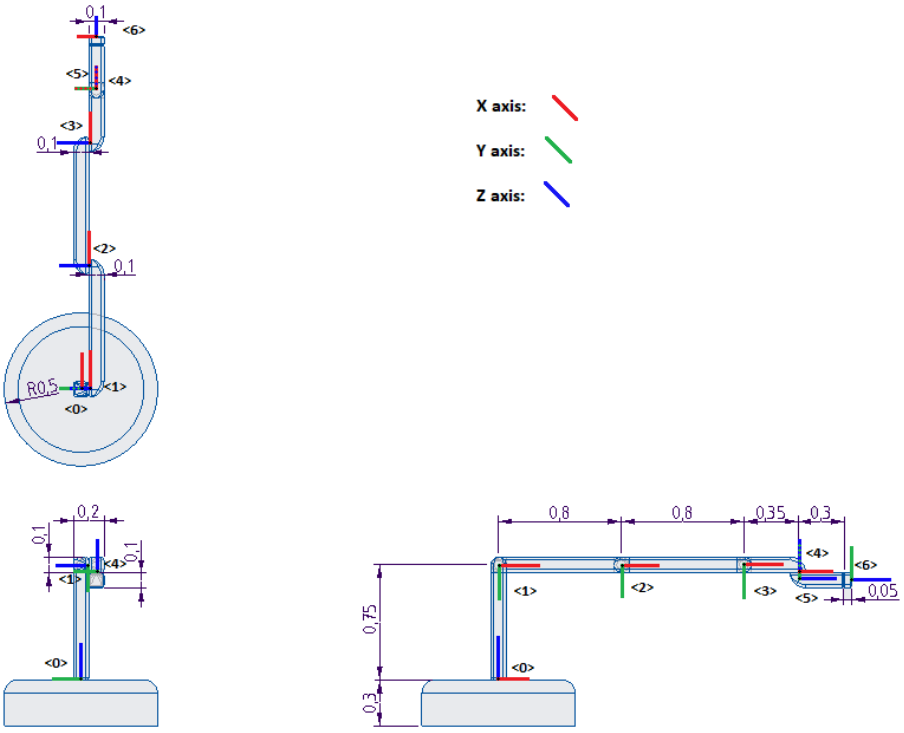

Figure 1: Schematics of the arm with frames highlighted

Frames	d_i	a_i	α_i	θ_i
frame1	0.75	0	$-\pi/2$	θ_1
frame2	0.05	0.8	0	θ_2
frame3	0	0.8	0	θ_3
frame4	0	0.35	$\pi/2$	θ_4
frame5	-0.05	0	$\pi/2$	$\theta_5 + \pi/2$
frame6	0.35	0	0	θ_6

Table 1: Denavit-Hartemberg table.

Apart from the first and last (which are cyclic) all joints in the arm have a range of motion of 300 degrees. The maximum speed is the same for all joints and is set at $45deg/s$. All the schematics and models of the manipulator can be found respectively in the schematics and models folders of the repository. In particular my_arm3D is the

.obj file of the whole arm. This is not used by the solution, however, thanks to the windows10 default tool *3D viewer* it can be opened to observe a three-dimensional representation of the proposed manipulator.

Figure 2: 3D represantation of the manipulator

2.2 Pick and place logic

The pick and place logic was implemented through a Finite State Machine (FSM) in the child script of the arm. There are five phases of the FSM:

- **State0: Grab object in the input conveyor**
  It computes a straight path from the pose of the tip of the manipulator to the target dummy in the object to grab. The condition for enter in this state is to have detected something with the proximity sensor in the input conveyor, instead, the condition for passing to the next state is to have grasped the part. This is tested by checking the parent of the anchor.

- **State1: Take object to the corresponding out conveyor**
  A vision sensor has been mounted on the end effector of the arm, once the

object has been grabbed, the sensor will be centered on it, by just checking the color of the part (this is a simulation of a computer vision algorithm) the manipulator will recognize the piece and a smooth path to the corresponding output conveyor will be computed. The condition for passing to the next state is to be at less than a cm to the target position.

- **State2: Approach the out conveyor**
  This state is very simple, it just computes a short vertical straight path in order to approach the conveyor and assure to have a more gentle drop. Also here the condition for passing to the next state is to be at less than a cm to the target position.

- **State3: Drop the object**
  In this state the piece is dropped. To do that, the anchor point is unlinked from the object. The condition for passing to the next state is to have waited one second from the unlink of the anchor.

- **State4: Get back to start pose**
  In this state, a smooth path from the current pose of the tip to the start pose of the manipulator is computed. The successive state is the state0, once more the condition for completing this phase is to be at less than a cm to the target.

3 Analysis and Optimization of the cell

3.1 Ability to supply multiple vendors at the same time

Since the spawning of parts is random and made in such a way that it is not possible to generate two consecutive identical parts, the simulation already guarantees (statistically) that the number of pieces over time will tend to be the same. To be more sure of this, three vision sensors have been added at the end of the outgoing conveyors. The *controlOutConveyors* script uses these vision sensors to count (and control) all parts and adjust the speed of each conveyor accordingly. In particular, a default and minimum speed of the outgoing conveyors is set, then, in parallel with the counting of objects (done in the sysCall_sensing), the speed of each conveyor is increased proportionally with respect to the difference with the conveyor with more pieces. In this way the conveyor with more objects will go at minimum speed while the other two will try to catch up.

3.2 Fully automated solution

The cell is fully automated and no human operator is required to complete its tasks. The area is enclosed with glass panels that allow viewing of the operations. The only way to enter the cell is through the door, which in the simulation is represented by the panel that leaves the cell open, note that this is done only to highlight the door, in a real implementation, obviously, the door would be closed while the robot is running, and the cell could only be accessed when the manipulator is still.

3.3 Flexible in terms of volume and mixed product handling

The solution guarantees high levels of flexibility both in the handling of volumes and mixed products. In the following sections these points are better explained.

3.3.1 Flexibilty in volume

The simulation is set to a slow default speed to allow a better understanding of all processes and make easier the debugging. The system is been tested also with higher speeds and responded well without problems. However, if you want to increase the speed a small tuning process is needed. The parameters to tune in particular are: the speed of the target dummy on the path, the speeds of the conveyors and the rate of spawning of the parts. The tuning is necessary because, of course, if you increase too much the speed of the conveyors and the rate of the spawn with respect to the speed of the trajectory, the manipulator won't be able to keep up. In a inverse situation, instead, the cell would not be efficient, because the fast manipulator would spend a big part of its time waiting that the parts reach the proximity sensor of the input conveyor. It is trivial, but worth noticing, that even by keeping good proportions between the three parameters, the system can not reach speeds beyond certain limits. For instance, because of the impossibility for the joints to reach too high speeds (remember that the upper velocity limit of each joint is set to $45deg/s$) but of course, there are many other reasons.

3.3.2 Flexibilty in mixed product handling

The suction pad of Coppelia is able to grasp any detectable object therefore even changing drastically the parts won't bother the simulation. In a real implementation of course this doesn't work. The suction cup in the real world needs a regular surface to be able to properly grasp an object, as long as the parts that need to be grasped have that surface, the system should have no problem, however, it would probably require computer vision algorithms to locate the right spot to take the object.

3.4 Zero-defect product

As already mentioned in section 3.1 each outgoing conveyor has a vision sensor for piece counting. The same vision sensors are used to verify that each part is in the correct conveyor. This problem does not occur in the simulation as the computer vision algorithm which recognizes the pieces is simulated by the color control alone, so it is very unlikely that the arm in Coppelia will bring the pieces into the wrong conveyors. In a real context, this problem is much more important, in fact, not only the computer vision algorithm of the arm may fail but it could also be that the parts at the end of the conveyors are damaged, in this case the system should be able to notice it. To verify that this function works properly in the simulation I suggest to manually put the pieces in the wrong conveyors. The system will warn you with a print in the Coppelia terminal that will tell you that an error has occurred and that you have a piece in the wrong conveyor. Also note that the wrong parts are not counted on the piece counters.