Implementation of a Virtual Box for Controlled and Effective Human-Robot Interaction

Jeacy Espinoza

Advisor: Prof. Marynel Vazquez

Background

A future where robots and humans frequently interact is inevitable and will hopefully be less contentious than the present day, where hitchhiking robots are beheaded and security robots are assaulted. Possibly due to fear, misunderstanding, or seeing robots as outsiders, people have been shown to act aggressively and violently towards robots in public spaces. A person’s tendency to harm a robot in public can only be resolved by making robots publicly available and studying the resulting behavior, hopefully leading to a more harmonious future between humans and robots.

Although Yale is filled to the brim with driven, sophisticated, and mature individuals, there will undoubtedly be more than a handful of those who will try to harm the cute robot Shutter from the Interactive Machines Group laboratory. Despite this sad reality, the benefits that would arise from positioning Shutter around Yale campus are great. These include curbing time-consuming and expensive laboratory experiments, as well as observing more natural human behavior due to the realistic setting where the human-robot interactions would take place.

Abstract

In order to make this a possibility, there would have to be a way to stop the subject from hurting both themselves and Shutter while also not making Shutter feel too distant. To achieve this goal, a protective box encasing Shutter could do the trick, but it would physically
separate Shutter from the subject making Shutter feel distant and causing the interaction to feel less authentic. The box would have to be virtual and capable of protecting Shutter and the subject by stopping Shutter’s movement if any object were to enter Shutter’s vicinity. Not only would this protect Shutter and the subject from scenarios in which a curious appendage may cause harm, but it would also make Shutter feel present to the subject and allow for better interactions.

**Procedure: Design and Implementation**

![Visualization of Shutter, the frame that encloses it, and the sensor that is positioned on top of the frame.](image)

Above is a visualization of Shutter, the frame that encloses it, and the sensor that is positioned on top of the frame. The virtual box’s walls are also the frame’s walls, and they share the same corners. It should be noted that Shutter’s position relative to the sensor may move, but it can never be close enough to the edges of the frame so that its movement can touch the virtual box’s walls, as will be explained soon.

The tools that were used to design such a virtual box involved Robot Operating System (ROS) and PCL (and ROS_PCL). The first steps that were taken were learning about both as well
as how to use PCL in ROS. PCL, which stands for “point cloud library,” was the primary way to deal with incoming information from the sensor positioned atop the frame that housed Shutter. ROS, was the primary way to communicate to the robot Shutter.

The initial approach was to voxelize Shutter and stop its movement once foreign voxels were detected by the sensor, but, ultimately, an alternative method was decided on. Taking advantage of the rectangular shape of the frame and the fact that at its resting position, Shutter’s head and body movement will never touch the “walls” of the frame, we decided to stop Shutter’s movement whenever we detected the presence of voxels in an individual face of the frame.

To accomplish this goal, the following steps were taken: first, we recorded about half a minute of sticking our hands into the frame and leaving the frame alone and stored it in a ROS bag. Next, taking Shutter’s position into account, I created a coordinate frame to represent the sensor’s position relative to Shutter when we were recording. Afterwards, using the depth data that the sensor collected in that bag, we generated a point cloud. I was then able to visualize this point cloud using a ROS tool called Rviz. Next, I down sampled that point cloud using a voxel grid filter from the PCL library and then published the result to a ROS topic for later access. This was also visualized using Rviz, where I could see my incoming hand from the sensor’s RGB camera turn into a collection of spaced out points. Next, I used the sensor’s coordinate frame to publish another coordinate frame in a corner of the frame encasing Shutter and then transformed the down sampled point cloud to that bottom corner. Next, I parsed the transformed point cloud to look for points with x, y, and z-coordinates that were a certain x-distance, y-distance, and z-distance from the corner. The x-distance was set as the
width of the frame, the z-distance was set as the height of the frame, and the y-distance represented a depth distance. Together these three parameters made up the three-dimensional volume where the program would test to find any points.

The very last step would involve calling on the service that stops ROS’s motors from functioning if a point is found entering the frame, as well as debugging, and making every program written more generalizable.

Next Steps

Now that the code has been written to create a ROS node that detects a foreign object entering the frame that encases Shutter, the program must either be modified so as to simultaneously work for the other faces of the frame that encases Shutter or make a more general program that accepts a face’s coordinate frame and then performs the appropriate transformations, voxelizations, and detections.