**Robot Code Documentation**

The way the code is built is that each behavior is programmed in its own function. As a result, they can be easily called by the main function when the robot is required to fulfill a certain task. At the same time, it is possible to stop any one behavior in case a behavior with a higher priority, such as keyboard input when the robot is wandering around, to take over. This functionality is similar to the layered control system described in Brooks’ paper. That is to say, behaviors that have a higher role in the layer can subsume lower levels whenever necessary.

Because each behavior is designed as its own function, and subscribers are designed to listen to them individually, there may need to be some communication between these behavioral modules such that the priority levels are adhered. That is, one module should be able to send a suppression message to another, and that subscriber will be able to hear it and take the appropriate course of action henceforth. For example, the robot must stop before turning – in the drive function of the code, once the distance of one meter is reached, driving is suppressed and the turn function activates once it is published. It is in this way that modules can communicate and suppress each other when the need arises.

This architecture follows Brooks’ subsumption architecture. These different modules and functions behave as layers, which, as mentioned before, have different priorities. These priorities, however, are mentioned in Brooks’ paper as “levels of competence.” In this code, the zeroth level of competence is handled with the drive and turn functions in order to produce our “wander” module. The level with the most precedence is the keyboard function, which uses teleop to determine what keystrokes drive the turtlebot. Because it has a higher precedence, it takes over the wander module. In this way, the control system is formed.

The code itself utilizes several global variables that function as a sort of “switch” for each module. These Boolean values determine whether a function should be running or if it should be repressed. For example, the keyboard function turns off many of these switches (in effect, repressing the other modules) by setting these values to false so that the teleop commands can be read. It also ensures that these values return to their true state when the keyboard is no longer in use. The turn function also utilizes this concept as it has to suppress the drive function. In a sense, when talking about the subsumption architecture in reference to these functions instead of modules as a whole, the drive function would be level zero.