Potpourri

• Software Design Review - In designing IMDL robots you will develop three types of software, mainly, low-level, sensor and motor routines
  – All behaviors are (should be) independent and can be both developed and debugged separately
  – We want to be able to add, remove or change behaviors at will without having to substantially rewrite software
  – We want to integrate behavior interaction easily

• Low-Level Routines
  – Perform housekeeping functions within the robot
  – Do not perform logic or decision-making
  – DO perform basic tasks in support of higher-level processes (e.g., read a digital compass and return orientation value)

Figure 1: Behavior Program Architecture
Potpourri

• Sensor Routines
  – Sensor control (e.g., controlling the frequency on an IR cannon mounted on a servo)
  – Collect data and fill global variable slots for essential external communication
  – Error correction
  – Default values
  – Filtering

• Motor Routines
  – Independent of behaviors (No behavior ever controls the motors directly). Behaviors send motor values to the motor routine via global variables
  – Facilitate smooth motor control
    • Non-smooth behavior may be caused by two or more behaviors competing for control
    • You may wish to implement a “smoothing” function on the motors for more natural “non-jerky” behavior. Jerky behaviors are not physical or biological and cause mechanical difficulties.
    • Switching motor direction or large speed changes can cause “spikes,” high currents and place high demand on battery packs
Potpourri

• Speed Control
  Speed = \( \frac{K}{K+1} \) Old_value + \( \frac{1}{K+1} \) New_value \((K \geq 0)\)
  Speed = \( \frac{K-1}{K} \) Old_value + \( \frac{1}{K} \) New_value \((K \geq 1)\)

Potpourri

• Collecting Sensor Data
  – Make an array, say DEBUG[20] and use it to record 20 readings of a global sensor value, etc. Now you can print all 20 values with a single `printf` statement.
  – Smooth the sensor data
    If you have, say, 20 values stored in S[20] then let
    \[ S_{\text{now}} = \frac{1}{20} \sum_{i=1}^{20} S[i] \]
  – Filter and/or correct the sensor data
Programming Behaviors

• Behavior Control
  – *All* behaviors run concurrently
  – The higher priority behavior should subsume the lower priority behavior
  – Each behavior is programmed in its own module
  – A function is written to perform behavior arbitration
  – Behaviors use the sensory global array of values to produce a desired set of global motor values

• Behavior Arbitration
  – Determine the dominant behavior and set the motor values to the appropriate level
  – May turn off unwanted or turn on additional behaviors
  – May modify the behaviors via global control variables
  – May adjust behavior priority

• Keep your behaviors relocatable
• Make sure you write code to test sub-systems of your robot as you develop them and keep those routines handy during the entire design process