Program design for a real time movement generator on an Arduino

Introduction and Goal

In order to make movements, a stream of parameters representing curiosity and level of excitement is streamed to the Arduino controller (which controls air pressure to the artificial muscles) at a faster rate than the mechanics of the robot can handle. It is therefore advantageous to be able to follow the required settings real-time as new data arrives, while adapting the movements that are being made to the actual value of the controlling parameters.

This document describes the approach to how this done.

Overall design

Ideally parallel process take care of reading new data on the one hand, and executing the resulting movements on the other hand. However, an Arduino has a single core, and parallelism can therefore only be simulated. The following considerations apply:

1. Processing power is a lot faster than the mechanical movements and hence program time scale is very different from movement time scale.
2. Internal timers, that generate a software interrupt can be used on the Arduino to start routines that handle some aspect of the software such as pressure monitoring.

It therefore stands to reason to:

1. Create a main loop that calls routines that only perform quick software actions, when their allotted time has arrived
2. To do monitoring of actual pressure of the muscles (and take appropriate action) using a timer interrupt. As a complete movement takes in the order of 4 seconds an interrupt that checks the pressure every 0.02 seconds seems to be a very good approach (experiments may show the need for a different timing).
3. An example of how to set up a timing routine for an Arduino nano is given here: <https://www.instructables.com/Arduino-Timer-Interrupts/>
4. An emergency timer interrupt running much faster to check if overpressure occurs can be made available. This can run at faster than 0.02 seconds

This leads to the following program design:

Setup {

Initialize timer

Initialize timings for actions

}

MainLoop {

If readInstructionTime and no emergency: {

Update instruction execution time

If Instruction available: {

Read instruction from PC

Set pressure and speed of movement Parameters accordingly

}

}

If (current-pressure lower than threshold){

Reset emergency state

}

}

MotorControlInterrupt {

If (current-pressure lower than setPressure) {

Increase Air pressure at preset rate

}

Else {

Decrease air pressure at preset rate

}

}

EmergencyInterrupt {

If (current-pressure higher than maxAllowedPressure){

Open emergency valve.

Set emergency state

}

}