Program design for a real time movement generator on an Arduino

Author: Erwin Engelsma

Date of last update: May 27 2022

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
5. For the current hardware implementation of the Arduino Nano the timer 1 cannot be used as its pins are also used as PWM outputs for motor control. Therefore the mainloop is run where a the time is checked and for the different routines when their time is up they are run.

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

}

}

Volume control

Originally it was thought that a pressure sensor can be used to check pressure and hence the level of extension of the silicone muscle. However, it was found that while the silicone muscle is in its true elastic mode this works, but when the actual movement of the muscle starts it goes into a non linear mode where pressure remains roughly constant as expansion takes place.

This lasts until a side of the silicone starts weakening so pressure actually goes down upon expansion. Therefore, the expansion should be controlled using the volume of air being pumped in.

This requires either a measurement of the air volume being pumped in (expensive sensor needed) or a form of calibration.

It will be experimented to see if the two main points of change in behavior (alin2nonlin: from linear behavior to nonlinear behavior and nonlin2reduced: from nonlinear behavior to reduced pressure upon expanding) can be used to come up with a relevant volume calibration. It must be taken into account that the motor behavior from PWM values (Pulse Width Modulation) is probably not linear. Hence calibration must be done from a defined start point to a defined end point.

alin2nonlin and nonlin2reduced are the obvious points for this.

The setup to do this is as follows, controlled from the Arduino:

* While experiment running
  + Initialize PWM value and sample time (given from Python program in msec)
  + Empty the silicone muscle to atmospheric pressure
  + Set valves for pumping action
  + Start the pressure pump at the given PWM (and set start time)
  + While nonlin2reduced not reached:
    - Read pressure
    - Send pressure, time (milliSec since start run) combination to Python
  + Stop pump and open valves
  + Send ‘measurement completed’ message to Python

On the Python side:

* Start Arduino with PWM value and sample time
* Prepare Excel file for data storage
* For the given set of PWM values:
  + Send requested PWM value to Arduino
  + Read datastream that comes back until measurement is completed
  + Store datastream in Excel file
    - header: PWM and sample time
    - data: Pressure and time of sample (in msec since start of run, so all measurements can easily be compared)