How to Use the *Delayline* class

Dennis Goldschmidt

May 18, 2012

AMOSII final motor neuron outputs are wired to the motors applying a so-called **delay line mechanism**. In this mechanism, the signals of the PSN and VRN's output neurons are send out to the motors with a certain time delay τ, respectively. To do so, we implemented a class named *Delayline*. This class contains functions to write signals into a memory buffer and to read out signals with a certain delay τ.

Constructor: The constructor creates a memory buffer vector $\vec{b}$ with a fixed size a. This size depends on the total required delay Γ: $a = \Gamma + 1$.

Write function: In the write function, the present signal $\sigma(t)$ is fed into the buffer vector $\vec{b}$ at the position i: $b_i = \sigma(t)$.

Read function In order to read out delayed signals, the read function returns the buffer vector value at the position $(i - \tau)$ mod a. Here we apply the modulo operation because $\vec{b}$ is used as a so-called **circular buffer** (see Fig. 1).

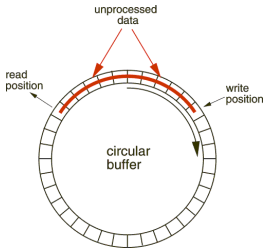

Figure 1: Concept of a circular buffer. The step length of the unprocessed data corresponds to the delay τ.

Step function The step function basically increments the step counter i and sets i equal zero after the last vector component is reached (circular buffer).

Usage in the controller file In order to use a *Delayline* class object in your controller, at first you have to call the class constructor in the beginning (**not** inside the step function, see Fig. 2). The parameter included in brackets is the size a of the buffer vector $\vec{b}$ (remember: this size depends on the total required

```
214
215 //Delayline constructor
216 tr_delayline = new Delayline(2 * tau + 1/*size of delayline buffer*/);
217 tl_delayline = new Delayline(tau_l + 2 * tau + 1);
218 ctr_delayline = new Delayline(5 * tau + 3);
219 fti_delayline = new Delayline(5 * tau + 1);
220 ftim_delayline = new Delayline(5 * tau + 1);
```

Figure 2: *Delayline* constructor.

delay Γ: $a = \Gamma + 1$). Here we have different sizes of the constructed *Delayline* objects and a delay $\tau = 16$ time steps ($\tau_{left} = 48$ time steps).

Secondly, you use the class functions (pointer of object) to write into and read out signals from your delay line. The write function should be called first and writes the value of the given parameter into the current step position of the buffer vector b_i (here output signals of the motor neuron network are written into the delay line, see Fig. 3).

```
381 /*****************************************************************************
382  *  MODULE 3 WIRING
383  *****************************************************************************/
384 //efficient delay line wiring (using delay line function)
385
386 //writing values into delayline buffer
387 tr_delayline->Write(tr_output.at(2));
388 tl_delayline->Write(tl_output.at(2));
389 ctr_delayline->Write(cr_output.at(2));
390 ftim_delayline->Write(fr_output.at(1));
391 fti_delayline->Write(fr_output.at(2));
392
```

Figure 3: Write function example.

You able to access any previous value up to the size of the delay line. Use the read function for a given delay τ and you get the delayed value (see Fig. 4).

In the end, you have to call the step function (see Fig. 5).

```
//read out delayed values

//TC joints
for (int i = TR0_m; i < (TL2_m + 1); i++) {
  if (i < TL0_m) {
    m_pre.at(i) = tr_delayline->Read((TR2_m - i) * tau); //Right side
  } else {
    m_pre.at(i) = tl_delayline->Read((TL2_m - i) * tau + tau_l); //Left side
  }
}

//CTr joints
for (unsigned int i = CR0_m; i < (CR2_m + 1); i++) {
  m_pre.at(i) = ctr_delayline->Read((CR2_m - i) * tau + (CR2_m - i));
  if (postcrold.at(i) > postcr.at(i)) {
    m_pre.at(i) = -1; //postprocessing
  }
}
for (unsigned int i = CL0_m; i < (CL2_m + 1); i++) {
  m_pre.at(i) = ctr_delayline->Read((CL2_m - i) * tau + tau_l + (CL2_m - i));
  if (postclold.at(i) > postcl.at(i)) {
    m_pre.at(i) = -1; //postprocessing
  }
}


//FTi joints
for (unsigned int i = FR0_m; i < (FR2_m + 1); i++) {
  m_pre.at(i) = fti_delayline->Read((FR2_m - i) * tau);
}
for (unsigned int i = FL0_m; i < (FL2_m + 1); i++) {
  m_pre.at(i) = fti_delayline->Read((FL2_m - i) * tau + tau_l);
}
//postprocessing
m_pre.at(FR1_m) = -1.2 * ftim_delayline->Read(tau - 1);
m_pre.at(FL1_m) = -1.2 * ftim_delayline->Read(tau + tau_l - 1);
m_pre.at(FR0_m) *= -1.5;
m_pre.at(FL0_m) *= -1.5;
m_pre.at(FR2_m) *= 1.5;
m_pre.at(FL2_m) *= 1.5;
```

Figure 4: Read function example.

```
//take one step
tr_delayline->Step();
tl_delayline->Step();
ctr_delayline->Step();
fti_delayline->Step();
ftim_delayline->Step();
```

Figure 5: Step function example.