

Spatial-Based ILC for Motion Control Applications

For presentation at the Second International Congress on Mechatronics (MECH2K3), Graz, Austria, July 14-17, 2003

Kevin L. Moore and YangQuan Chen
Center for Self-Organizing and Intelligent Systems (CSOIS)
Utah State University, UMC 4160
Logan, UT 84322-4160
email: moorek@ece.usu.edu

Abstract

For many motion control applications spatial constraints are often more important than temporal constraints. In recent work we have developed a spatial control strategy called the ϵ -controller for mobile robot applications. The control strategy is based solely on static path geometry with position (in space) feedback. Motivated by this idea, in this paper we consider the notion of spatial-based iterative learning control. Specifically, we consider repetitive operation problems where corrections are made to the control signal trial-to-trial. Unlike traditional ILC, however, which updates control signals based on the time elapsed along a trajectory, we instead make updates based on path errors and progress along the path. The idea is demonstrated via simulation for a system with bang-bang velocity control.

1 Iterative Learning Control

Iterative learning control, or ILC, is an approach to improving the tracking response of systems that are operated repetitively [1, 2]. By “repetitively” we typically mean that some type of finite-length trajectory is to be executed over-and-over, starting each execution, or trial from the same initial condition. For instance, consider Figure 1. Here we imagine a robot arm in an assembly line. At the start, the arm is at rest. Once a workpiece is in place the arm is commanded to move to some point in space and perform an action (e.g., a spot weld). It then returns to rest and waits until another workpiece is in place. This process repeats *ad infinitum*.

ILC is concerned with improving the performance of the system (e.g., the robot arm motion in Figure 1) from trial-to-trial. Typically, ILC performance is measured as a function of the error along the trajectory, where it is always implicitly understood that the trajectory is parameterized as a function of time. Thus, for the example in Figure 1, at trial k each joint an-

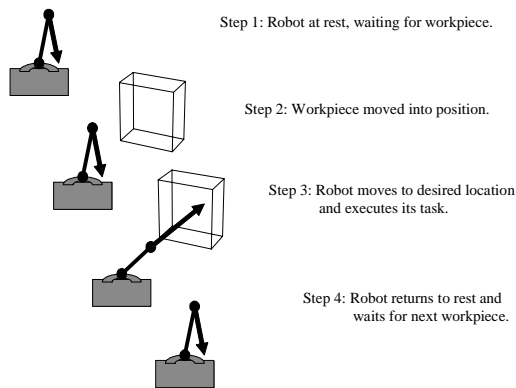


Figure 1: ILC idea.

gle $\theta_{ik}(t)$ would have an associated reference trajectory $\theta_i^d(t)$ (a triangle function in this example) and the joint torque $\tau_{ik+1}(t)$ at trial $k+1$ would be updated according to an algorithm of the form:

$$\tau_{ik+1}(t) = \tau_{ik}(t) + f(\theta_i^d(t) - \theta_{ik}(t))$$

for each value of time $t \in [0, T]$, where T is the (fixed) trial length. In this equation the function $f(\cdot)$ is called the “learning algorithm.” The design and analysis of ILC learning algorithms is relatively well understood in this time-based formulation (the references above can be consulted for more details on ILC and ILC learning algorithms).

2 Path Tracking versus Time Tracking: The ϵ -Controller

Path tracking problems are common in robotics and other motion control applications. A system is often required to follow a path in space with a certain speed in order to complete a certain mission. The desired path is often parameterized in time to provide time-varying position set-points. Since the locus of desired set-points follow (in time) the trajectory (in space), it

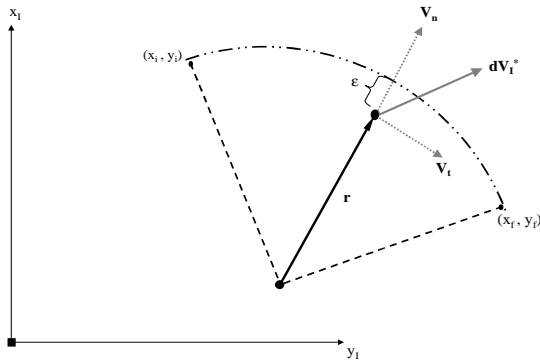


Figure 2: ϵ -Controller quantities.

is inferred that a controller that can track the parametric set-points in time will effectively track the desired trajectory in space. Because this is an indirect approach, system dynamics can generate unexpected results (e.g., in the presence of external disturbances or in cases of actuator saturation). In particular, saturated actuators may cause the system to “cut corners.” Thus, if in a particular application the requirement of “following the path” is more important than the position of the WMR along the path at a given time, then a different approach should be used which is time independent and is completely spatial. Working in the context of wheeled mobile robots (WMR), we developed a novel control law which, while maintaining the WMR on the path in space, impels it toward the desired endpoint of the path with the desired speed [3, 4]. This path tracking controller (dubbed the ϵ -controller) is based completely on static inputs - the geometry of the path - and the desired speed along with feedback of the current position of the WMR.

The essential idea of the ϵ -controller as developed initially in [3] is shown in Figure 2. The dashed arc represents the desired path and we assume the desired speed (V_d) is parameterized along the path (in general it can be allowed to vary continuously along the path, though more commonly it is taken as a constant). We assume the WMR is at the location represented by the vector \mathbf{r} from the center of the arc. As shown in the figure, ϵ is the (closest) distance to the path from the WMR’s position. If we assume the arc representing the desired path has radius R , then the error is

$$\epsilon = |R| - \|\mathbf{r}\|$$

We assume the WMR can “drive” to generate a velocity vector through suitable kinematics computations and low-level actuation and control. Let $d\mathbf{V}_1^*$ (the superscript “*” denotes setpoint) be the sum of the normal and tangential velocity components of the WMR, \mathbf{V}_n and \mathbf{V}_t , respectively. The job of the *epsilon*-controller

C_ϵ is to generate the vector ($d\mathbf{V}_1^*$) that specifies corrective motion as a function of deviation from the path. This is done as follows. The normal velocity component \mathbf{V}_n is the primary corrective action taken by the C_ϵ to minimize ϵ . Thus C_ϵ conceptually acts as a regulator operating on ϵ . This regulator could consist of any of several control methods. For the P control case, for example, we have

$$\mathbf{V}_n = K_p \epsilon \left(\frac{\mathbf{r}}{\|\mathbf{r}\|} \right)$$

where K_p is the proportional controller gain. Multiplication by the normalized vector \mathbf{r} gives the desired *direction* of \mathbf{V}_n (we assume computations are done in the inertial coordinate system). In the tangential direction, we define the normalized vector \mathbf{V}_t to be

$$\mathbf{V}_t = (V_d - \|\mathbf{V}_n\|) \left(\frac{\mathbf{r}_t}{\|\mathbf{r}_t\|} \right)$$

where the quantity $(V_d - \|\mathbf{V}_n\|)$ is never allowed to be negative. This choice of constraints makes $\|\mathbf{V}_t\|$ a function of ϵ . The result is a tangential velocity which approaches zero as ϵ increases and approaches V_d as ϵ goes to zero. Finally, ($d\mathbf{V}_1^*$) is obtained as the (normalized) vector sum of the normal and tangential velocity vectors. More details of this strategy can be found in the above-mentioned references. The key concept is the use of deviation from the geometric path as a control parameter rather than using the time-based trajectory error.

3 A Spatial ILC Strategy

In this section we present via a simulation example a spatial ILC strategy. We consider the following decoupled, second-order, relative degree one plant (where t is an integer and k is the trial index):

$$\begin{aligned} x_k(t) &= 1.9x_k(t-1) - 0.9x_k(t-2) + u_k^x(t-1) \\ y_k(t) &= 1.9y_k(t-1) - 0.9y_k(t-2) + u_k^y(t-1) \end{aligned}$$

Our goal is to track, in space, the path shown in Figure 3. We assume that the control inputs on each axis are limited to be on-off, bang-bang controls with a fixed velocity $v = 0.3$. Our goal is to drive the system to exactly follow the path with no spatial overshoot and to finish at a final time $t_f = 100$. That is, each trial length will be 100 time steps. However, it should be stressed that we do not use time to determine our control input. Instead, we use progress along the path to determine when the control is on or off. Progress along the path, denoted $Pr \in [0, 1]$ is determined by finding the closest point from the actual path to the desired path and then computing the percentage completion of the total path length based on that closest point.

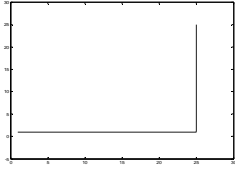


Figure 3: Desired path for example.

Once progress has been determined our control signals are turned off and on according to the following logic, shown in pseudo-code:

```

if ( $Pr_k(t-1) < x_{off}$ )
  Set  $u_k^x(t-1) = v$ 
  Set  $u_k^y(t-1) = 0$ 
else
  if ( $Pr_k(t-1) < y_{on}$ )
    Set  $u_k^x(t-1) = 0$ 
    Set  $u_k^y(t-1) = 0$ 
  else
    if ( $Pr_k(t-1) < y_{off}$ )
      Set  $u_k^x(t-1) = 0$ 
      Set  $u_k^y(t-1) = v$ 
    else
      Set  $u_k^x(t-1) = 0$ 
      Set  $u_k^y(t-1) = 0$ 
    end
  end
end
end
end

```

Basically, the idea is that along the path we turn the velocity in the x and y directions on and off, as defined by the variables x_{off} , y_{on} , and y_{off} (for this example, given the geometry of the desired path, we can *a priori* pick $x_{on} = 0$ and $y_{on} = 0.5$ by default. The distinction is that the on and off decisions are made based on progress rather than time elapsed (although time t appears as an argument of Pr in the pseudo-code, this simply means the progress at that time and is not used as a decision trigger; the decision is based on the comparison of progress (at that instance) to a threshold).

The next piece we add is the ILC algorithm. This is used to update the switching variables after each trial as follows:

$$x_{off}^{k+1} = x_{off}^k + \gamma_x \max_{t \ni Pr > 0.5} (25 - x_k(t))$$

$$y_{off}^{k+1} = y_{off}^k + \gamma_y \max_{t \ni Pr \geq 0.5} (25 - y_k(t))$$

That is, we look at the maximum spatial overshoot after reaching the half-way point along the path and use this to adjust when we turn off the x drive component. Likewise, we look at how much y -axis spatial overshoot occurs after we have come even with the end of the path and use this to adjust when the y drive component is turned off. While admittedly ad hoc for this example, the idea can be extended to more general problems.

The effectiveness of this algorithm for this example is illustrated in Figure 4, which shows the path outputs of the first, second, and eighth trials, respectively on the same graph with the desired path. By the eighth trial the system follows the path exactly, ending at the desired time. Figure 5 shows the control inputs for the first and eighth trial. In the first trial we arbitrarily (though motivated by path geometry) set $x_{off} = 0.5$ and $y_{off} = 1$. As can be seen in the figure, the algorithm successfully changed the values of these variables to eliminate spatial overshoot. We see that x_{off} is reduced, thus reducing the speed of the system so that it just comes to a stop and does not overshoot. Notice that, from the perspective of time, the y -axis drive is turned on much later in the eighth trial than in the first trial. This is because the overall velocity along the x -axis is reduced so as eliminate the x -axis overshoot (i.e., we don't drive so hard), thus it takes longer to reach 50% progress. It should also be noted that the spatial ϵ -controller would develop a similar profile, though it would still have spatial overshoot, but it would not necessarily do it in the same trial length (i.e., it might take longer). The distinction here is that we are making the improvements on a trial-to-trial basis.

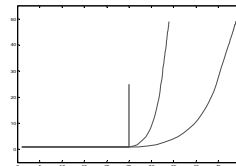


Figure 4: Example results: first, second, and eighth trials.

4 Conclusions

In this paper we have presented some initial results on the marriage of the notion of spatial-based control with that of iterative learning control. Motivated by the previously-developed ϵ -controller, we demonstrated via example a trial-to-trial adjustment of switching variables in a bang-bang, constant velocity system being forced to follow a path in a fixed time. The distinctive feature of the ideas is that of switching based on progress along the path, rather than on elapsed time. A number of other issue remain for further development, including:

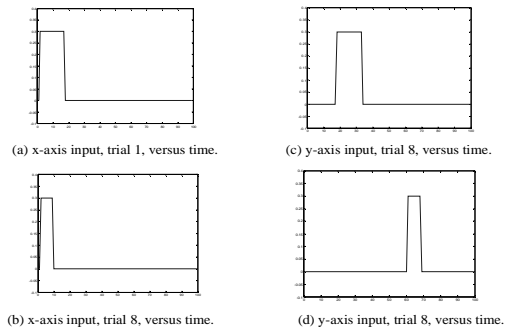


Figure 5: Control inputs.

1. Generalization to more sophisticated paths, including arcs.
2. Generalization to adaptation of all switching variables (in our example two of four were fixed).
3. Generalization to variable velocities (this combined with item two implies the ability to do learning for pulse-width modulated types of systems).
4. Generalization to non-bang-bang control.

We are currently working on these ideas as well as their implementation using a pointing system comprised of a high-resolution, two axis gimbal with visual measurement of paths.

References

- [1] K. L. Moore, “Iterative learning control - an expository overview,” in *Applied and Computational Controls, Signal Processing, and Circuits*, vol. 1, pp. 153–214, March 1999.
- [2] K. L. Moore, *Iterative Learning Control for Deterministic Systems. Advances in Industrial Control*. Springer-Verlag, 1993.
- [3] M. Davidson and V. Bahl, “The scalar ϵ -controller: A spatial path tracking approach for ODV, Ackerman, and differentially-steered autonomous wheeled mobile robots,” in *Proceedings of IEEE Int. Conference on Robotics and Automation*, (Seoul, Korea), pp. 175–180, IEEE, 2001.
- [4] M. Davidson, V. Bahl, and K. L. Moore, “Spatial integration for a nonlinear path tracking control law,” in *Proceedings of American Control Conference*, (Anchorage, Alaska), May 2002.