A Distributed Real Time Coordination Protocol

Lui Sha[1] and Danbing Seto[2]
[1]CS, University of Illinois at Urbana-Champaign
[2]United Technology Research Center

Abstract: When the communication channels are subject to interruptions such as jamming, the coordination of the real time motions of distributed autonomous vehicles becomes a challenging problem, that differs significantly with fault tolerance communication problems such as reliable broadcast. In this paper, we investigate the issues on the maintenance of the coordination in spite of arbitrarily long interruptions to the communication.

1 Introduction

Internet based instrumentation and controls are an attractive avenue for the development and evolution of distributed real-time systems [1, 2]. However, one of the challenges is the real-time coordination problem in the presence of communication interruptions. In distributed control, coordination concerns with how to synchronize the states of distributed control subsystems in real-time. A prototypical problem is to command a group of unmanned air vehicles, where each vehicle must closely follow a desired trajectory which is planned in real-time.

To synchronize the states of distributed control systems, a reference trajectory[1] is given in real time to each distributed node, a local system. A reference setpoint[2] moves along the reference trajectory according to the specified speed profile. The reference trajectories are designed in such a way that the movements of the reference setpoints represent the synchronized changes of distributed states. The difference between the actual system state and the state represented by the reference setpoint is called tracking error. A tracking error bound specifies the acceptable tracking error on each reference trajectory. A local controller is designed to force the local system's state to follow the reference setpoint closely within the tracking error bound.

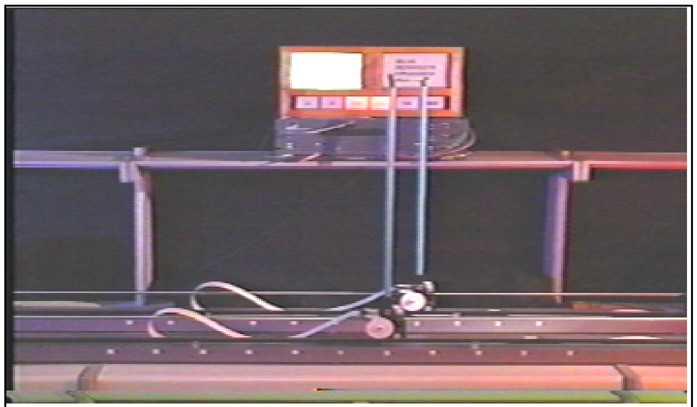

Figure 1: Coordinated IP Control Prototype

[1] A reference trajectory is a specification of how a system's state should change as a function of some independent variables. For example, the glide slope that guides aircraft landing specifies both the path and the speed along the path.

This paper addresses the abstract problem of designing reliable communication protocols for distributed real-time coordination with a concrete example. We will use a simplified Inverted Pendulum (IP) control prototype to introduce the basic concepts in real-time coordination.

The coordinated IP control system has two IPs with a nearly massless plastic rod which ties the tips of the two IPs together as shown in Figure 1. The rod does not affect the inverted pendulum control until the slack is consumed by the difference in IPs' positions. Each IP consists of a metal rod mounted vertically to a hinge on a motorized cart controlled by a computer. The metal rod rotates freely. It will fall down from its upright position if the cart's movement is not properly controlled. The mission of the overall system is to get the two IPs moving in synchrony to a desired position on the track with the IPs standing upright. Apparently, if the two IPs are significantly out of steps with each, they can pull each other down. Therefore, the two carts must keep the pendulums at upright position, and maintain their positions synchronized within a small tolerance of, e.g., *5 cm*, to prevent the plastic rod from falling. The tolerance is a function of how tightly the two tips are tied together.

In this experiment, each IP is controlled locally by a computing node on an Ethernet switch. An operator uses a Command Node on the network to send messages commanding the two IPs where to go. A "communication jamming station" is also connected to the same network, so that we can experimentally test the robustness of communication protocol designed for coordinated control.

Example 1: As illustrated in Figure 2, suppose that in a coordinated IP control experiment, the initial positions for the two IPs are at the middle of the two parallel tracks, i.e., $x_1 = 0$ *cm*, $x_2 = 0$ *cm* at time $t = 0$. We may command the IPs to move to positions near one end of the parallel tracks at *70 cm* with a motion start time $t = 10$ *sec*, and with a constant speed of *2cm/sec*. The system coordination is carried out by sending commands to the IPs. A command specifies both the start time and the reference trajectory**.** A reference trajectory specifies the path of the motion and

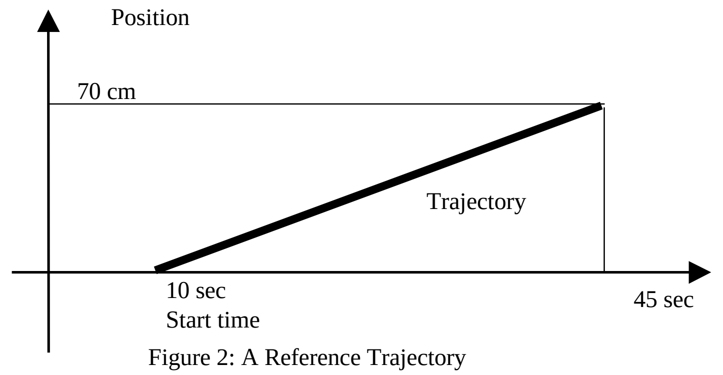

Figure 2: A Reference Trajectory

the speed along the specified path. In this example, paths for the IPs are two straight lines, each connecting track position 0 to track position 70 cm.

Suppose that both IPs receive their commands in time, that is, before the start time $t = 10$ *sec*. The Local Reference Setpoint will start moving exactly at $t = 10$ *sec* and with a constant speed of *2 cm/sec*. The local control forces the IP to follow the Local Reference Setpoint**.** If both IPs' con-

2 Giving a reference trajectory, the reference setpoint specifies where the controlled system's state ought to be along the reference trajectory. That is, it is a specification of the desired system state as a function of time. Feedback control is used to force the physical system's state to converge to the reference setpoint.

trols are functioning correctly, the tracking error between the IP position and the Local Reference Setpoint will tend to zero. In the experiment, the error between the two IPs positions is allowed to be as large as *5 cm*, which is called as the global coordination error bound. Typically, in distributed control systems, the global coordination error bound is translated into sufficient conditions that can be observed and controlled locally. For example, if each IP is within *±2.5 cm* of its Local Reference Setpoint, then the global coordination error bound is satisfied. This localized condition is referred to as the local tracking error bound.

In Example 1, both IPs start their motions at the same time. In practice, it is quite common to command different objects starting motions at different specified times. However, distributed real-time coordination with synchronized start times is the key problem. The problem of using different start times can always be decomposed into two problems: 1) a coordination problem with synchronized start times and 2) a stand alone control problem. For example, suppose that initially one of the IPs, IP_1 is at – *5cm* while IP_2 is at *0 cm*. We would like them to line up first and then move in synchrony. This problem can be decomposed into two problems: 1) command IP_1 to first move to $x_1 = 0\ cm$, and 2) command them to move in synchrony as illustrated in Example 1. Obviously, the hard problem is coordinated control with synchronized start times. In the following, we will focus on problems that require synchronous start times.

We have so far assumed that both IPs receive their commands on time. This assumption is unrealistic in an open network. Obviously, if one IP receives its command on time, and the other receives its command much later, the IP that moves first will pull down the other IP. This is an example of coordinated control failure. The design of the real-time coordination communication protocol concerns with the problem of how to send the trajectories to distributed nodes quickly and reliably. That is, in spite of arbitrary long interruptions to any or all of the communication channels, the protocol must guarantee that distributed nodes will never receive a set of inconsistent commands that will lead to coordination failure.

This problem is related to the synchronization of distributed objects [3] in the sense that the states of distributed objects cannot be diverged arbitrarily. However, in the synchronization of distributed objects, the problem is how to force distributed executions to enter a set of prescribed states when certain condition is met, not how to quickly and reliably communicate the trajectories that constrain state transitions.

The communication protocol design for real-time coordination is similar to the design of fault tolerant communication protocols, such as reliable broadcast [4, 5], in the sense that we need to find a way to reliably provide distributed objects with consistent information. However, in real-time coordination, we have a weaker form of consistency constraints due to the existence of tracking error bound. On the other hand, we are faced with a hard constraint on the relative delays between the messages received by coordinating nodes. We will revisit this point after we specify the real-time coordination problem. In Section 2, we will define the problem and show some of the pitfalls in protocol design. In Section 3 we present the solutions. Section 4 is the conclusion and summary.

2 Problem Formulation

In this section, we will define the communication protocol design problem for real-time coordination. Our assumptions are as follows:

Assumption 1: Communication delays change widely and unpredictably. They are normally short with respect to application needs. However, very long delays can happen suddenly without warning.

Assumption 2: Messages are encrypted. Adversaries are unable to forge or alter the content of messages without being detected.

Assumption 3: The clocks of distributed nodes are synchronized.

Assumption 4: The control of objects is precise. Errors due to control algorithms, environments or mechanical problems are negligible.

Assumptions 3 and 4 allow us to ignore tracking errors due to control or clock synchronization inaccuracies. That is, they allow us to focus on the specific problem of tracking errors caused by communication delays. The local tracking error bound in the following discussion is used only to constrain the tracking error due to communication delays. From a system engineering perspective, this is the portion of the tracking error bound that is allocated for tracking errors due to communication delays. It is important to note that the bound for control errors is in the form of ± **B**, because the object being controlled can either undershoot or overshoot the reference setpoint. The tracking error due to late start is always positive. It measures how much the object lags behind the reference setpoint. We will use the symbol **B** to denote the tracking error bound in the rest of this paper.

Assumption 5: A reliable point-to-point communication protocol is used in all the communications.

In coordinated control (with synchronous starts), we could specify fixed start times as in Example 1. However, we cannot guarantee the coordination to work using fixed start times, if the duration between start time and current time is shorter than the worst case communication delay. Observing this constraint causes long delays to the communication of the trajectories. We are interested in protocols using start times that are set dynamically to take advantage of the window of opportunities in communication – moments at which bandwidth is available.

The simplest coordination protocol using dynamic start time is to let each node start its motion immediately after it has received its command. Although this simple-minded protocol will not work in the presence of arbitrary delays, it helps us to pin down a number of useful concepts. The idea of dynamic start times is to use some communication protocols to dynamically start the motions within a narrow time window. To analyze the worst case relative delay in start times, the System-Start-Time (SST) is defined as the leading edge of the time window. That is, the time at which one of the coordinating nodes makes the first move.

To compute the local tracking error due to late start, we imagine that the Local Reference Setpoint starts at SST, independent of the time at which the local coordination command is received. This is, what the Local Reference Setpoint should have done if there were no delay. We call this idealized setpoint the "SST-Reference Setpoint", and call the trajectory "SST-Reference Trajectory". The tracking error due to late start is then computed as the difference between SST-Reference Setpoint and the actual position of the object due to late start.

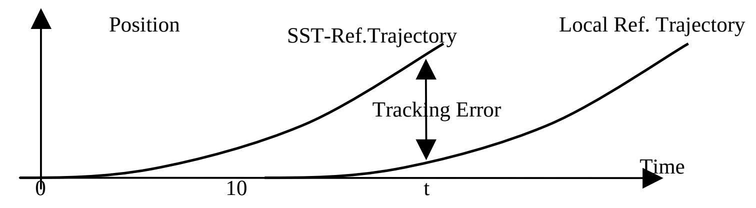

Figure 3: Tracking Error Using Dynamic Start Times

Example 2: Let the system start time SST = 0. But the node starts its motion 10 sec later due to the delay in receiving its coordination command. The SST-Reference Trajectory and the Local Reference Trajectory are illustrated in Figure 3. Note that the physical object follows the Local

Reference Setpoint to move along the Local Reference Trajectory. At time *t*, the local tracking error due to late start is the difference between the positions of SST-Reference Setpoint and the Local Reference Setpoint. Note that the position of Local Reference Setpoint and the position of the object is identical under Assumptions 1 to 5. That is, the object follows the local reference setpoint perfectly. The tracking error in Figure 3 is caused by the late start of the object. Next, we illustrate some of the difficulties in the design of real-time coordination communication protocol using dynamic start times.

Example 3: The Command Node sends a command to nodes N_1 and N_2. When N_1 (N_2) receives its command, it immediately sends N_2 (N_1) a confirmation message that it has received its command with the given command identifier. When N_1 (N_2) receives its command and the confirmation from N_2 (N_1) that N_2 has also received the corresponding command, N_1 (N_2) starts its motion immediately.

Unfortunately, the protocol in Example 3 does not work. Suppose that N_1 receives its command and the confirmation from N_2 at time *t*, and therefore starts its motion at time *t*. Unfortunately, N_2 receives N_1's confirmation message long after time *t*. Therefore, the large lag leads to coordination failure.

A moment's reflection tells us that the consensus based dynamic start time protocol is not better than fixed start time protocol. It is not possible for distributed nodes to reach a strictly consistent view of a given command within a duration shorter than the maximal communication delay *D*. To see this point, note that in any consensus protocol, there will be a decision function, which will return the value *True*, if a certain set of messages are received. All the adversary needs to do is to let one of the coordinating nodes receive all the required messages, and start its motion. However, the adversary jams the required messages to other nodes for duration *D*. Indeed, if we insist on finding a way to ensure that all the nodes receive the same set of commands, it becomes a reliable broadcast problem. It is not possible to guarantee reliable broadcast within a time window that is less than the worst case communication delay. Fortunately, the real-time coordination problem permits a weaker form of consistency due to the existence of tracking error bounds.

3 Protocol Design

There are two requirements for the design of communication protocols for real-time coordination. First, under a given communication protocol, the tracking error due to late start must always stay within its bound no matter how long is the communication delay. Second, it is desirable to shorten the time that the protocol needs to send the reference trajectories to coordinating nodes.

As a first step to exploit the weaker form of consistency, we develop the Constant Distance Grid (CDG) – Iterative Command (IC) protocol.

A CDG partitions each reference trajectory into a series of k equal distance short segments. Each segment on a trajectory should be no longer than the tracking error bound. Figure 4 illustrates a simple Constant Distance Grid with two parallel reference trajectories. Position 0 on each trajectory marks the starting point of the first segment, Segment 1 of the trajectory.

Given a trajectory in the form of CDG, the Iterative Command (IC) protocol works as follows. The Command Node sends messages to each of the N Maneuver Control Nodes and asks them to move to Position 1 first and wait for further commands. Once the Command Node receives messages that all N Maneuver Control Nodes have reached Position 1, it commands them to move to Position 2 and wait, and so on. The IC protocol is outlined by the following pseudo-code.

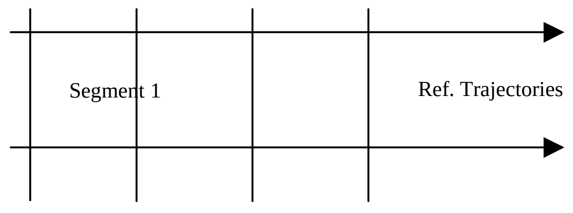

Figure 4: A Constant Distance Grid

Definition 1: The IC Protocol

```
Initialization:
Each of the N objects will be at its starting position (Position 0) of its trajectory.

Command Node
for j = 1 to k      // k is the last position, the final destination for a trajectory.
  Send Message j to each of N Maneuver Control Nodes to go to Position j;
  Wait for confirmation of reaching Position j from all the N Maneuver Control Nodes;
end

Each Maneuver Control Node
Loop
   Wait for command;
   Move to the commanded Position j;
  Send confirmation to the Command Node
         immediately after reaching the commanded Position j;
end Loop
```

The CDG-IC protocol is just the IC protocol that uses CDG. We now analyze the tracking error. Recall that the worst case tracking error is computed with respect to the SST-Reference Setpoint, which starts to move as soon as an object makes the first move. Let the segment length be d and the tracking error bound be $\boldsymbol{B}$ and $d \leq \boldsymbol{B}$.

Theorem 1: Under CDG-IC protocol, the local tracking error, e, on Trajectory i is bounded by the tracking error bound, $\boldsymbol{B}$.

Proof: Suppose that Theorem 1 is false, i.e., $e > \boldsymbol{B}$. Since the segment size $d \leq \boldsymbol{B}$, we have $e > d$. For $e > d$, SST-Reference Setpoint and the object must be in two different segments. Let the object be in Segment i and the SST-Reference Setpoint at Segment j, and $j > i$. For SST-Reference Setpoint in Segment j, a command it must receive a command to go to Position $j+1$. Under CDG-IC protocol, a command to go to Position $j+1$ will be issued only if all the objects have reached Position j, the starting position of Segment j. This contradicts the assumption that the object is in Segment i and the SST-Reference Setpoint in Segment j. Theorem 1 follows.

CDG-IC has the drawback of waiting for all the objects to complete the current command before issuing the next one. However, the movement of electronic messages is much faster than the movement of physical objects. Waiting for the movement of physical objects could waste the window of opportunities in communication.

To speed up the process of sending the trajectories, we have developed Constant Time Grid - Fast Iterative Command (CTG-FIC) protocol. There are two key ideas in CTG-FIC protocol.

- To replace the constant distance grid with a constant time grid. In a constant time grid, the distance of a segment is adjusted in such a way that each segment will take the same time to finish with respect to a giving reference trajectory.
- To send commands to objects to go to Position *(j+1)* without actually waiting for all the objects actually reaching Position *j*.

As soon as the Command Node receives all the acknowledgements from all the objects that they have received the command to go to Position *j*, it sends messages to command them to move to Position *(j+1)* until the command for the final destination position is successfully received. In order words, we allow an arbitrary number of outstanding, yet to execute commands. This allows us to capitalize on the windows of opportunity in communication. Due to the lack of space, we are unable to show the proof of correctness of the GTG-FIC protocol. Readers who are interested in knowing some of the potential pitfalls in designing fast protocols that allow outstanding commands or interested in the proof of CTG-FIC may send emails to the authors to request for a copy of the report: "Communication Protocols for Distributed Real-Time Coordination in The Presence of Communication Interruptions."

4 Summary and Conclusion

Internet based instrumentation and controls are an attractive avenue for the development and evolution of distributed control systems. However, one of the challenges is the design of communication protocols for real-time coordination in the presence of communication interruptions.

Two protocols were developed to solve the real time coordination problem in the presence of communication interruptions, the Constant Distance Grid - Iterative Command protocol (CDG-IC) and the Constant Time Grid – Fast Iterative Command protocol (CTG – FIC). Both of them can tolerate arbitrary long communication delays without causing coordination failures. However, the completion time of sending a trajectory to a node under CDG-IC depends on the speed of physical systems. CTG-FIC can send successive commands to distributed nodes without waiting for the completion of the earlier commands. Thus, the completion time of CTG-FIC is independent of the speed of the physical systems. It can better exploit the window of opportunities in communication. Due to the limitation of space, only the simpler CDG-IC is described and the CTG-FIC was briefly outlined.

Acknowledgement

This work was sponsored in part by the Office of Naval Research, by EPRI and by the Software Engineering Institute, CMU. The authors want to thank Michael Gagliardi, Ted Marz and Neal Altman for their contributions to the design and implementation of the demonstration software and to thank John Walker for the design and implementation of the hardware. Finally, we want to thank Jane Liu for her helpful comments to an earlier draft.

References:

1. The Proceedings of NSF/CSS Workshop on New Directions in Control Engineering Education, October, 1998. pp.15-16.
2. The Proceedings of Workshop on Automated Control of Distributed Instrumentation, April, 1999.
3. J. P. Briot, R. Cuerraoui and K. P. Lohr, "Concurrency and Distribution in Object-Oriented Programming", ACM Computing Survey, Vol. 30, No. 3, September, 1998.
4. P. M. Melliar-Smith, L. E. Moser and V. Agrawala, "Broadcast Protocols for Distributed Systems", IEEE Transaction on Parallel and Distributed Systems, January, 1990.
5. P. Jalote, "Fault Tolerance in Distributed Systems", Prentice Hall, 1994.