Development of a Distributed Multi-rate Motion Control System Using USB

Sungsoo Rhim
Department of Mechanical Engineering
Kyung-Hee University
Kyunggi-Do, 449-701, Korea
e-mail: ssrhim@khu.ac.kr

Soon-Geul Lee
Department of Mechanical Engineering
Kyung-Hee University
Kyunggi-Do, 449-701, Korea
e-mail: sglee@khu.ac.kr

ABSTRACT

This paper describes a PC-based distributed multi-rate real-time control system using USB protocol, which is developed as a general motion controller. The control system consists of two control programs: one running at 1 kHz sampling rate on a PC with Linux and another running at 10 kHz sampling rate on a remotely located motion control card called RASID (remote axis serial interface device). Two programs communicates through USB at every 1 msec. A USB communication driver is developed to ensured the 1 msec desired communication time. The main program running on the PC generates reference trajectory at 1 kHz and send it to the RASID through USB and RASIDs located near the motors gather the sensor information and execute the low-level control at 10 kHz. The USB-based connectivity reduces the wiring harness and eventually the manufacturing cost of the machine. The multi-rate nature of the developed system improves the control capability. The effect of sampling rate is analyzed and simulated.

I. INTRODUCTION

Although still there are many issues to be resolved, the use of the standard PC as a real-time control platform is an irresistible universal trend [1], [2], [3], [4]. So far the use of the standard PC as a real-time control platform has been mostly based on the cost benefit. However with recent progress made in PC industry that statement quickly becomes obsolete. The use of the standard PC has not only economical benefits over the conventional expensive customized high performance control platform but also it now begins to gain ground for its competitive performance in accomplishing hard real-time requirements. Many researchers around the globe start to adopt or consider adopting the standard PC as their hard real-time control platform. With the availability of inexpensive and powerful DSP chips and improvement of real-time operating systems running on PC, PC-based real-time controllers have promising future for the industry use as well as the academic use. This paper is a report on the development of a novel standard-PC-based multi-rate real-time control system using USB communication protocol (refer to the attached image). The control system that we developed uses an open-architecture design that permits all aspects of the control implementation. The developed control system consists of a standard PC with Linux as a platform and a novel motion control card called RASID (Remote Axis Serial Interface Device) per axis. The RASID performs the function of motor control, taking armature encoder feedback, and causing the motion of the armature to follow a prescribed path as dictated by a 1 millisecond sample period command from the main controller (which resides on the PC). The main loop runs at 1 kHz on the PC and manages high level tasks, which includes the trajectory generation, feedforward control and other advanced control processes. The main loop on the PC and the low-level control loop on the RASID communicates using USB protocol at 1 kHz. The low-level control loop on the RASID up-samples the trajectory signal transmitted from the main loop and executes a 10 kHz conventional PID feedback control process. The use of USB protocol in communication allows us to mount the RASIDs near the amplifiers of the motor to be controlled throughout a machine. With this dispersed and distributed mounting of the RASIDs (generally one per axis of motion) the wiring harness, which usually takes significant portion of the manufacturing cost, can be greatly reduced. The multi-rate control approach adopted in our control system is to distribute computing load between the processors on the PC and the RASIDs and to achieve high control performance with less computing power. The smaller the sample period, the closer a digital control system will approximate a continuous-time system. An investigation included in the paper shows that the minimum error achievable using a discrete-time PD controller is inversely-proportional to the square of the sample period. Using the developed control system, we have performed a series of experiments to verify the point made by our investigation. Results from the simulation is included in the paper.

II. PC-BASED MULTIRATE REAL-TIME CONTROL SYSTEM USING USB

A. Overview

In this research, we have developed a novel PC-based multi-rate real-time control system using USB protocol, which uses an open-architecture design that permits all aspects of the control implementation. The real-time control system consists of a motion control card called RASID (Remote Axis Serial Interface Device) whose picture is shown in Figure 1 per control axis and a PC that runs on the standard Linux operating system with a modified thread scheduler, a low-latency patch,

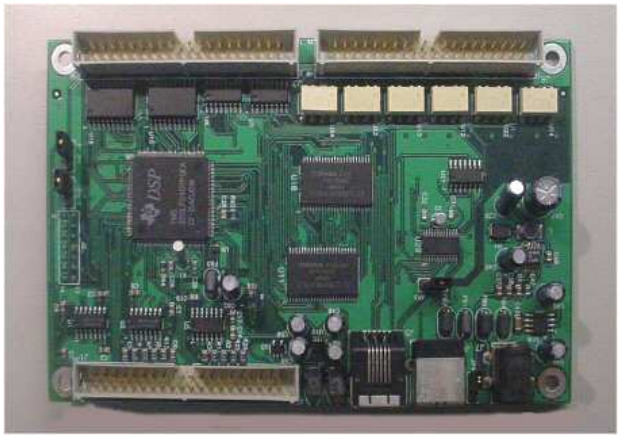

Fig. 1. Picture of RASID

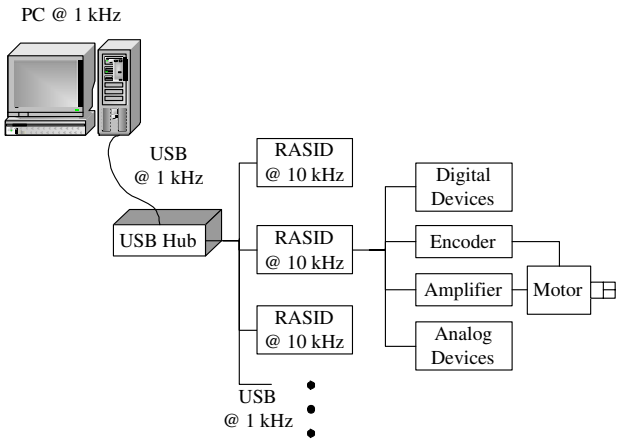

Fig. 2. Configuration of distributed multi-rate real-time control system using RASID

and a pre-emptive patch [6]. In the control system two control loops are executed. Figure 2 shows the configuration of the PC and RASIDs. The main loop running at 1 kHz on the PC manages high level tasks, which includes the trajectory generation, feedforward control and other advanced high-level control. The communication between the two control loops is done using the isochronous USB protocol [5]. As illustrated in Figure 3, the main loop talks to the feedback loop on each RASID every 1ms through USB cable and it transmits the next reference position and velocity and the feedforward effort to each RASID. The feedback control loop on RASID receives the reference trajectory point and up-samples it to get 10 kHz reference trajectory. In return, the RASID transmits its current states to the main loop so that the main loop can update necessary process. Each RASID has a TI DSP chip, 2 encoders, 2 DAC channels, 6 ADC channels, 7 digital outputs including 2 FETs, 8 digital inputs, single RS-232 connection, and a USB communication chip. In the data packet that each RASID transmits to the main loop, the current states of DAC, DIO, RS-232 communication, encoder, PID controller are included. By looking into the status data packet from RASIDs the main loop finds out all the details of the machine states. The communication through USB cable between the two control loops allows us the dispersion of RASIDs. In other words we can locate RASIDs away from the PC and near the amplifier of its respective axis motor and all the sensors. Then all the sensor data can directly be fed into RASID and the RASID delivers the information to the main loop on the PC through only one USB cable. The dispersed and distributed configuration of our control system could significantly reduce the wiring harness and manufacturing cost consequently.

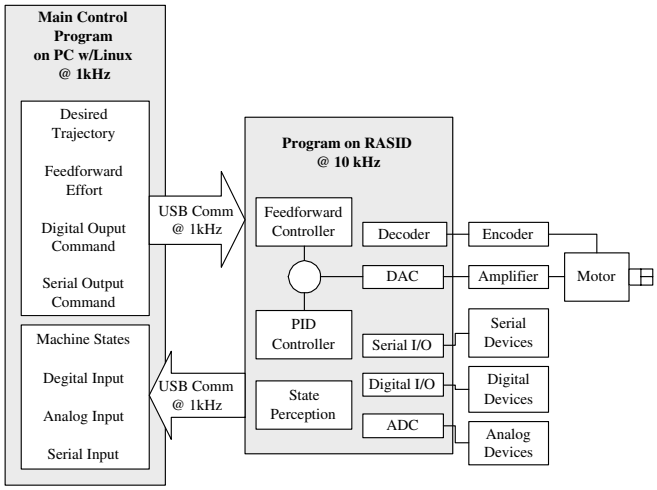

Fig. 3. Configuration of two programs: Main control program running on PC and RASID program

B. Communication components

The main control program communicates with the RASIDS attached to the machine by means of the Universal Serial Bus (USB) protocol.

There are essentially 5 parties that are involved in this communication.

1) Control program: reads data from and writes data to the attached RASID device(s) through the Linux Operating System
2) KHubD: the Linux daemon that is in charge of responding to physical connections to and disconnections from any USB port
3) Rasid Driver: program that knows how to communicate with the RASIDs and that keeps track of all connected RASIDs. In charge of communicating with the RASID at a high level.
4) Host Controller driver (HC): driver in charge of communication with the USB host controller that is integrated into the USB card or the motherboard chipset
5) RASID Device: reads data from Camcode and writes data to Camcode through the USB wire

C. Communication events

From the user perspective, there are seven major events that occur in order for communication between Camcode and any RASID device to occur.

1) Connection of the RASID device: This event occurs when a RASID device is physically connected to the USB port of the host computer. The USB host controller detects that a new RASID has been attached to the system, and KHubD is notified of this connection. KHubD next notifies the RASID driver that a new RASID has been connected by calling the driver's "probe" function. The RASID driver ensures that the connected device is a RASID device, and then sets up any data structures necessary for communication with the RASID.
2) Opening the RASID device: Before the control program running on the PC can write to the RASID, it must open the RASID file located in the /dev/ folder. Each connected RASID has a corresponding file named Rasid$x$, where x is a number 0 - 15. When the main control program opens this file, the RASID driver immediately initiates continuous communication with the RASID device that corresponds to that file. It does this by submitting data structures known as URBs (USB request blocks) to the HC driver which then queues all of the URBs. Each URB corresponds to exactly one write to or one read from the RASID. Each Write URB has a buffer that contains the data to send to the RASID, and each read URB has a buffer to save any data that has been read from the RASID. When the read or write transaction of a URB is completed, the Host Controller informs the RASID Driver of the completion, handing the RASID driver the old URB. In the case that the old URB is a write URB, the RASID driver copies the next message to be written to the RASID to the write URB's send buffer and resubmits the URB to the HC driver. In the case that the old URB is a read URB, the RASID driver copies the message that was received from the RASID into a buffer, and then resubmits the read URB for further reading.
3) Writing to the RASID device: When the main control program writes to an already open RASID, the RASID driver copies the data to be used into a buffer. The next time a write URB returns after sending data to the RASID, the RASID driver will copy this data into the write URB's buffer and resubmit the URB to the HC driver so that this new data will be sent to the RASID. If there is no new data to be submitted to the RASID, the URB is given a data block consisting of all zeros to send to the RASID and is resubmitted to the host controller driver.
4) Reading from the RASID device: When the main control program reads from an already open RASID, the RASID driver returns the data that exists in the read buffer. This data was copied into the read buffer, the last time a read URB was returned to the RASID driver by the Host Controller driver.
5) Closing the RASID device: After the main control program finishes using the RASIDs, it closes the open file that is associated with the RASID device. For every open of a RASID file, there must be a corresponding close of the file, after which, the RASID driver will stop the continuous resubmission of URBs to the host controller. At this point, there is no communication between the RASID and the main control program, or the RASID and the RASID driver.
6) Disconnection of the RASID device: This event occurs when a RASID device is physically disconnected from a USB port. The USB host controller detects that a previously connected RASID has been disconnected and notifies KHubD that this RASID has been disconnected. KHubD next notifies the RASID driver that this device has been disconnected by calling the "disconnect" function of the RASID driver. The disconnect function destroys all of the RASID's URBs and destroys all data structures that correspond to the old RASID. Any subsequent read from or write to the disconnected RASID will fail.

D. Communication interval

The main control program guarantees a data transfer of 1 kHz. This means that every millisecond, new data must be sent from Camcode to the RASID. To accomplish this, the RASID driver requests that the HC driver transfer one packet of data every millisecond. In addition, Camcode must create the next data packet for the RASID every millisecond without fail. To ensure that this frequency is maintained the RASID driver stores both the time that the main control program writes to the RASID and the time that the HC driver finishes sending the data packet to the RASID. Every millisecond, the main control program reads these values from the driver. It then checks the time difference between the main program writes and the time difference between USB transfers for latencies greater than one millisecond. If the delay passes a certain tolerance, the main program must stop. The operation log of actual execution of two control loops was recorded and the sampling timing of each control loop was examined. Experimentally measured communication intervals are plotted in Figure 4. Figure 5 shows the frequency of each communication interval. The plots show the latency of the USB communication is less than 5 μsec. The main control loop is triggered by the USB communication interrupt signal which occurs every 1 msec. The measured latency for the sampling interval for the main loop is less that 15 μsec.

III. Effect of Sampling Time

A. Analysis

The RASID, running at a ten-times faster rate than the main loop and thus able to update ten times as often, fills in the gaps between lower rate samples generated in the main loop by linear interpolation. A PID feedback control is wholly implemented on the RASID. The faster sample rate at the RASID is thus used to achieve tighter armature control than would be possible with the PC alone. To investigate the effect of the sampling rate, we investigate the stability and steady-state error of a rigid robot modeled as a double integrator with a one step delay under PD control as shown in Figure 6.

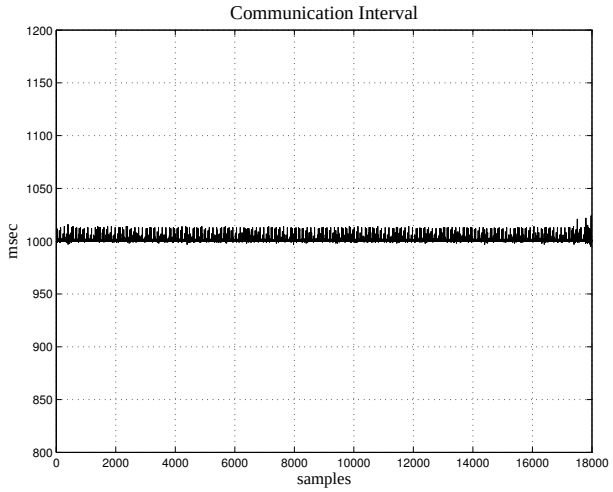

Fig. 4. Experimentally measured communication sampling intervals

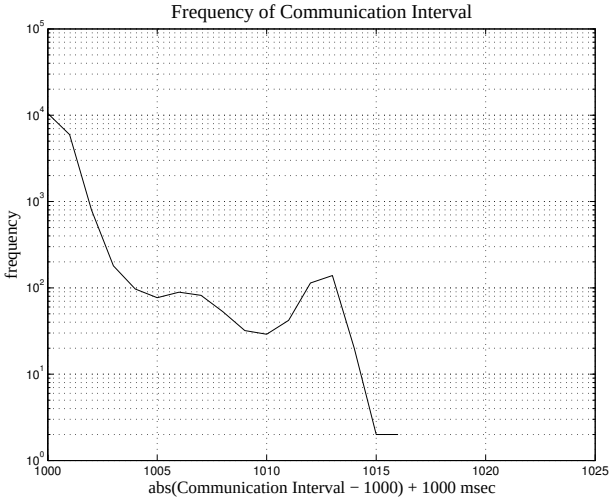

Fig. 5. Frequency of experimentally measured communication sampling intervals

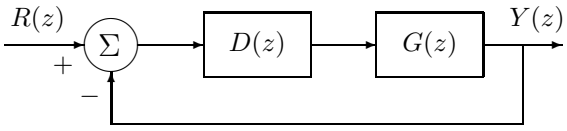

Fig. 6. Double integrator with one step delay under PD control

The transfer functions of this system and its PD controller are given by

$$G(z) = \frac{T_s^2}{2}\frac{z+1}{z(z-1)^2} \tag{1}$$

$$D(z) = K_p + K_d\frac{z-1}{z} \tag{2}$$

The resulting error transfer function is

$$\frac{E(z)}{R(z)} = \frac{z(z-1)^2}{z^2(z-1)^2 + (az-b)(z+1)} \tag{3}$$

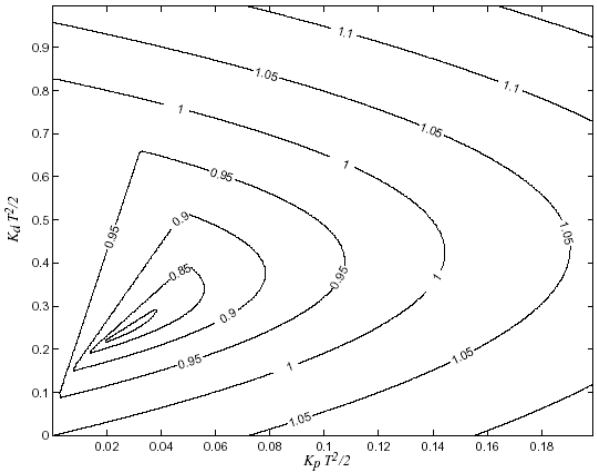

Fig. 7. Contour plot of stable regions

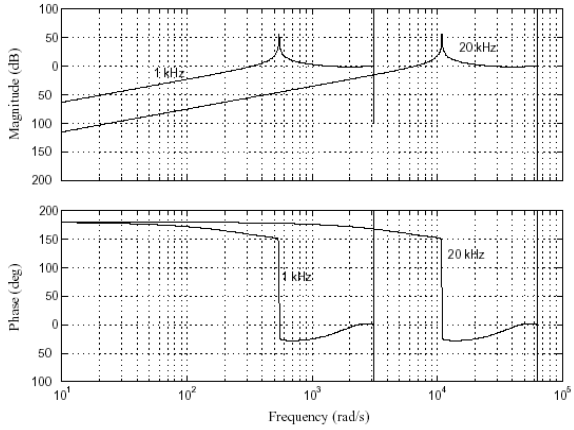

Fig. 8. Bode plots for error transfer functions for 1 kHz and 20 kHz sampling rates

where $a = (K_p+K_d)T_s^2/2$ and $b = K_dT_s^2/2$. The steady-state error at low frequencies is

$$\left|\frac{E(z)}{R(z)}\right| \approx \frac{\left|e^{j\omega T_s}-1\right|^2}{K_pT_s^2} \approx \frac{\omega^2}{K_p} \tag{4}$$

Thus the error is minimized by maximizing the proportional gain K_p, which is limited by the stability requirement of the system. A contour plot of the stable regions (levels less than one) is shown in Figure 7. From the plot, the largest proportional gain with an acceptable transient response is $K_p = 0.2/T_s^2$ resulting in the steady-state error amplitude of $|E(z)/R(z)| \approx 5T_s^2\omega^2$. Therefore, the steady-state error, which is an indicator of the tracking performance, is inversely proportional to the square of the sampling rate. The bode diagram of the error transfer function for sampling rates of 1 kHz and 20 kHz are shown in Figure 8.

B. Simulation results

For a simple mass which is modeled as a double integrator, we have implemented PD controller using 1 kHz sampling

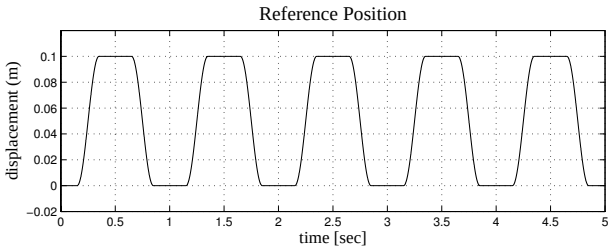

Fig. 9. Reference trajectory

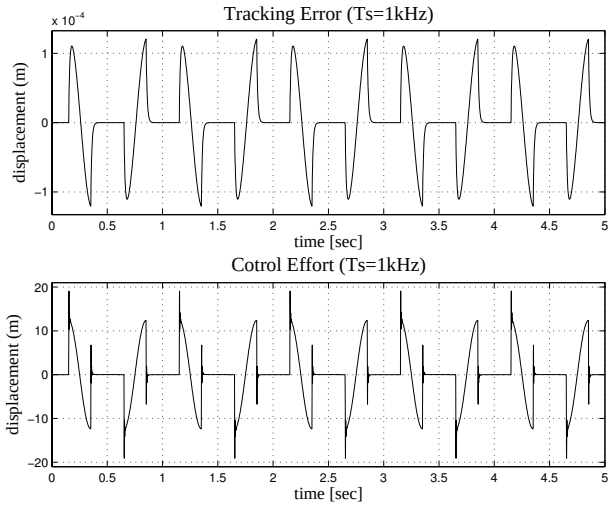

Fig. 10. Tracking error and control input for $T_s = 1$ kHz

rate and 10 kHz sampling rate and the control performances were compared. Figure 9 shows the repetitive reference input given to the control system. Figure 10 and Figure 11 shows the simulation results for the PD control system with 1 kHz sampling rate and 10 kHz sampling rate respectively. For each controller, the maximum stable control gains calculated in the previous subsection were used. As shown in the analysis the tracking error with 10 kHz sampling rate was much smaller ($1/10^2$ than that with 1 kHz sampling rate. The better tracking with 10 kHz was achieved without causing larger control effort. Note the scale difference in tracking error plots in the figures. The tracking error plot for the 1 kHz case shown in Figure 10 is in 10^{-4} scale and the tracking error plot for the 10 kHz case shown in Figure 11 is in 10^{-6} scale.

IV. Conclusions

In this paper we introduce a novel PC-based multi-rate real-time control system using USB protocol and investigate the characteristics of the developed control system. The merits of the control system are presented in analytical and experimental ways. The developed real-time control system consists of a standard PC with a modified Linux operating system and a motion control card called RASID (Remote Axis Serial Interface Device) per axis and. There are two control loops running in the devised control system. The one is the main loop runs at 1 kHz on the PC and the other is a low-level

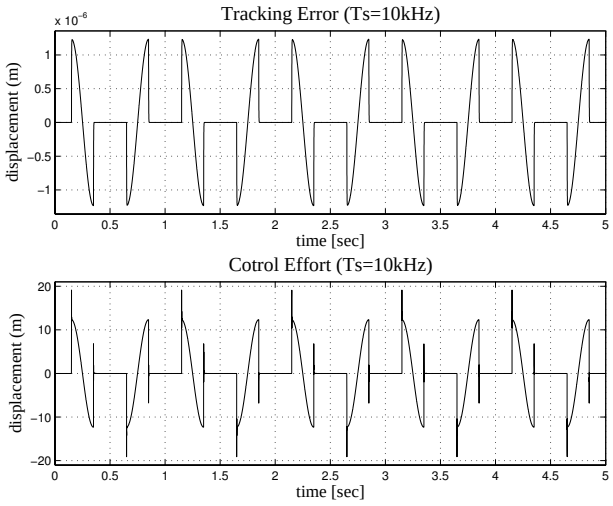

Fig. 11. Tracking error and control input for $T_s = 10$ kHz

feedback control loop runs at 10 kHz on RASID. The both loops communicate to each other using USB protocol at 1 kHz. The communication based on USB protocol allows us to locate the RASIDs near the motor throughout the whole machine and helps us to reduce lot of wiring harness and manufacturing cost consequently. The low-level control loop up-samples the trajectory signal transmitted from the main loop running on the PC and executes a 10 kHz conventional PID feedback control process. The multi-rate control approach adopted in our control system is to distribute computing load between the processors on the PC and the RASIDs to achieve high control performance with less computing power. This paper presents and discusses the effects of sampling rate on tracking error. An investigation shows that the minimum error achievable using a discrete-time PD controller is inversely-proportional to the square of the sample period.

References

[1] Feng, Xin, S.A. Velinsky, Daehie Hong, 2002 "Integrating Embedded PC and Internet Technologies for Real-time Control and Imaging," *IEEE/ASME Transactions on Mechatronics*, **7**, pp. 52–60.

[2] Lee, C.J., C. Mavroidis, 2001, "PC-based Control of Robotic and Mechatronic Systems under MS-Windows NT Workstation," *IEEE/ASME Transactions on Mechatronics*, **6**, pp. 311–321.

[3] Costescu, N., D. Dawson,M. Loffler, 1999, "QMotor 2.0-a Real-time PC-based Control Environment," *Control Systems Magazine*, **19**, No. 3, pp. 68–76.

[4] Yi, X., Y. Wenjun, Z. Jing, 2002, "Research on Robot Control System Based on PC," *Proceedings of the 4th World Congress on Intelligent Control and Automation*, **2**, PP. 1246–1250.

[5] Axelson, J., 2001,*USB Complete*, Lakeview Research, Madison, WI., USA.

[6] http://www.linux.org/,"Linux User's Manual".

[7] Nichols, B., D. Buttlar, J.P. Farrell, 1996, *Pthreads Programming*, O'Reilly.

[8] Maxwell, S.A., 1999, *LINUX: Core Kernel Commentary*, Coriolis Group Books.