

Algorithms and Software for Nanomanipulation with Atomic Force Microscopes

A. A. G. Requicha, D. J. Arbuckle, B. Mokaberi and J. Yun

Laboratory for Molecular Robotics
University of Southern California
Los Angeles, CA 90089-0781, USA
requicha@usc.edu

Abstract

Interactive manipulation of nanoparticles by mechanically pushing them with the tip of an AFM (Atomic Force Microscope) is now done routinely at many laboratories around the world. However, a human in the loop introduces significant inaccuracies and results in a very slow process, mostly because of the need for locating the particles before and after manipulation operations, in the presence of large spatial uncertainties which are often comparable to the size of the particles.

This paper describes the nanomanipulation systems developed at USC's Laboratory for Molecular Robotics over the last decade, culminating in a fully automatic system that is capable of accurately positioning small nanoparticles, with diameters ~ 10 nm. This system uses software compensators for the nonlinearities inherent in the piezoelectric actuators used in most AFMs. The planner and execution systems are described, as well as the software architecture of the systems. Experimental results are presented that validate the approach and show that nanoparticle patterns that would take hours to build interactively can now be built in minutes. Automatic operation makes possible the construction by manipulation of nanostructures much more complex than those built in the past.

1. Introduction

Scanning Probe Microscopes (SPMs) have provided a unique window into the nanoworld since their invention in the mid 1980s. They have remarkable imaging and measurement capabilities, which range from atomic-resolution topographical (i.e., height) images of surfaces, to density-of-states images, to measurement of electrostatic, magnetic and bond-breaking forces, to name a few. SPMs not only can probe materials but can also modify them at the nanoscale, and have been widely used by the nanorobotics community for manipulating nanoparticles, nanowires, nanotubes and other objects that may serve as building blocks for the bottom-up assembly of nanostructures.

Atomic manipulation was demonstrated soon after the invention of the SPM. It can provide new insights into nanoscale phenomena, but it does not seem very useful for building devices and systems at the nanoscale. Typically, atomic manipulation is done in UHV (ultra high vacuum) at temperatures near 0 K, it is too slow for engineering

applications since it relies on placing a single atom at a time to build a nanostructure, and has difficulties ensuring that intermediate constructions are stable.

Manipulation of nanoparticles or other objects with dimensions comparable to those of large molecules, at room temperature and in an ambient environment, is a more promising fabrication process. It is potentially useful for building prototype devices, repairing structures fabricated by other means, or constructing molds or templates for other nanofabrication processes such as nanoimprinting or templated self-assembly. The Atomic Force Microscope (AFM), which is a specific type of SPM, is normally the “robot” of choice for nanomanipulation, since it operates both on conductive or insulating surfaces, in air or in a liquid, and is relatively inexpensive. Building nanostructures by AFM manipulation compares favorably with other techniques such as electron-beam lithography in cost, repeatability, accuracy, and resolution. AFM robotics is akin to classical macroscopic robotic manipulation, as used, for example, in the automotive industries, but differs from it because the minute spatial dimensions at which the AFM operates have strong scaling implications. The dominant physical phenomena are different, and require new strategies and approaches.

Today, most software systems for AFM nanomanipulation are interactive. This is sometimes advantageous—for example for exploratory investigations. However, interactive operation is very slow and labor intensive for nanostructure fabrication. Automation is obviously desirable, but it has been elusive until recently.

We describe below, in Section 2, an interactive system developed at USC’s Laboratory for Molecular Robotics (LMR). This serves to introduce the major problems that lie on the path to automation. The remainder of the paper is devoted to the automatic planning and execution system designed and implemented at the LMR. The planner is described in Section 3. Next, the execution system, which compensates for the spatial uncertainties inherent in AFM operation, is introduced in Section 4. Section 5 discusses the software architectures of our implementations. Results are presented in Section 6, and conclusions are drawn in the final section. More detailed introductions to AFM manipulation and surveys of the research in the field are available in [Requicha 2003, 2008].

2. Interactive Manipulation

A typical approach to nanoparticle manipulation with the AFM proceeds in three steps. First, image the sample (or part thereof) to find where the particle is. Second, move against the particle along a trajectory that passes near its center, and change the AFM parameters from their imaging values so as to increase the force applied to the particle, thereby pushing it. Third, re-image to see where the particle ended up. At the LMR we have been for the past decade successfully using a protocol in which the two imaging steps above are done in dynamic force (i.e., “tapping”) mode, and pushing is accomplished simply by turning off the z feedback. Without active feedback the tip moves horizontally and runs into the particle, mechanically pushing it.

The imaging steps are needed because there are spatial uncertainties associated both with the imaging and the manipulation processes. If we image the sample and find a particle at some position (x, y) , and then instruct the AFM to move to that position, most of the time we will not find the particle there. With fixed voltages applied to the piezoelectric actuators used by most AFMs, the tip keeps moving continually with respect to the sample, as a result of thermal drift which is caused by slight fluctuations of temperature in a system composed of several materials with different coefficients of expansion. A full AFM image, typically 256x256 pixels scanned at a 1 Hz frequency, takes several minutes to acquire. During that time, the tip may drift on the order of 10 nm or more. The diameters of the particles we normally manipulate are 5-15 nm. Therefore, ignoring drift would result in missing completely a particle, and we need to search for the particle before pushing it—hence Step 1 above. In LMR's interactive system we draw a straight line segment on the image of the sample to command the AFM to scan along that line. Drawing several lines on the image near the approximate location of a particle and examining the resulting single-line scans, it is easy to estimate the actual position of the particle's center. Immediately pushing in the center's direction, without waiting for the drift to become appreciable, normally produces the desired result. The video in Extension 1 illustrates this process. A user moves a red line, maintaining it parallel to itself and parallel to the desired motion direction, near the estimated position of a particle, so as to find a scan with maximum particle height, which corresponds to the particle center. Then the user sets two points along the scan line for turning the feedback off and on, and instructs the AFM to execute the push command.

Drift is not the only cause of spatial uncertainty in an AFM. Creep and hysteresis also have non-negligible effects and must be taken into consideration when moving small nanoparticles. Creep effects are significant when a relatively large displacement of the tip takes place. The tip initially moves quickly to a value near the intended one, but then starts moving slowly and approaches the final position approximately along the tail of an exponential function, taking sometimes several minutes to reach a final value. Waiting for drift to end slows down the manipulation process in an unacceptable manner. Not waiting can cause errors on the order of tens of nanometers.

Hysteresis is another cause of spatial uncertainty. For example, scanning a particle along the same line in the forward and backward directions can produce images that differ by more than the diameter of the particles we normally manipulate.

Drift, creep and hysteresis are successfully combated in the interactive procedure outlined above and shown in the video. While searching for the particle, we always scan in the same direction, and move the probing line only slightly. This helps minimize hysteresis and creep. And once we find the center of the particle, we push it immediately, so that drift does not have time to grow significantly.

Why is Step 3 above needed? Figure 1 shows typical results of pushing 15 nm particles with our usual protocol. The horizontal axis is the commanded push distance, and the vertical axis is the distance actually traveled. It is clear that commanded and actual distances are substantially the same for motions below a few tens of nanometers, but the

process performance deteriorates for larger motions, on the order of 100 nm or more. Therefore, we have no guarantee that the manipulation has the desired result, especially for larger motions, and must search for the particle again after each push.

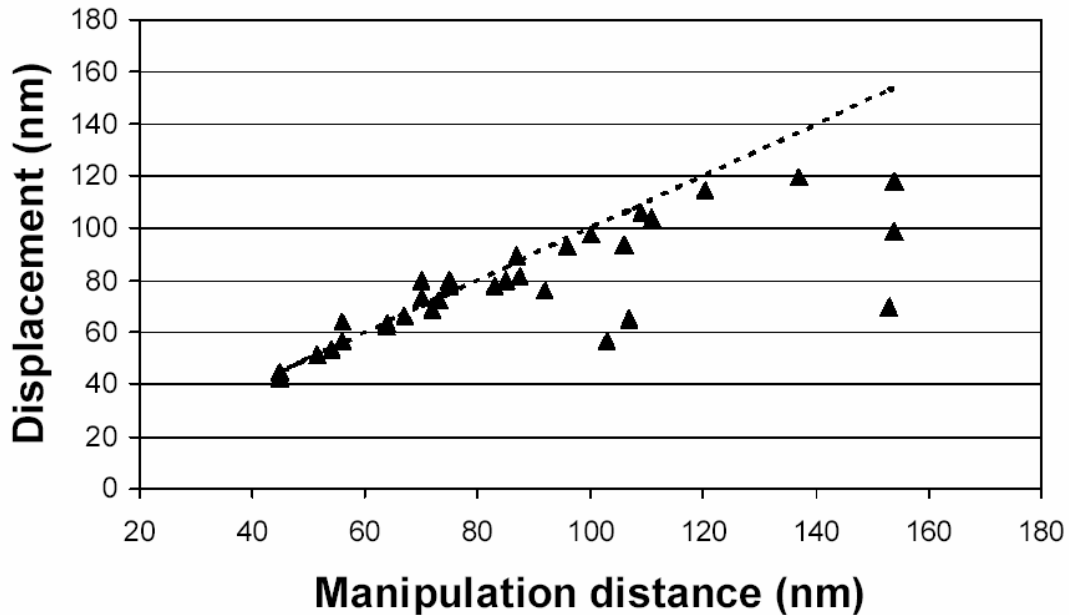


Fig. 1 – Reliability plot, showing (on the y axis) the actual distances traveled by 15 nm Au nanoparticles as a function of the commanded pushing distances (on the x axis). The dashed line corresponds to the ideal operation, when actual and commanded distances are equal.

All these search and correction operations, these latter necessary when the desired result is not achieved in one push, are very time consuming. It can take a whole day's worth of a skilled operator's time to build a typical pattern of some twenty nanoparticles of small diameters (10 nm or so). The spatial uncertainties discussed here pose a significant challenge for efforts aimed at automating the nanomanipulation process. As we will see below, one must compensate for the effects of these nonlinearities to be able to operate without a user in the loop.

3. Planning

The planning problem for automatic manipulation of nanoparticles may be stated as follows: given an initial configuration of nanoparticles on a surface (usually a random dispersion), and a goal configuration, find a sequence of positioning and pushing trajectories that, when executed by the tip, will convert the initial configuration into the final one. We solve this problem in several steps, discussed in the following subsections.

3.1 Matching

We begin by deciding which particle in the initial configuration should be moved into each target location in the goal state. We want to ensure that the total distance traveled by the tip during the pushing motions is as small as possible. This is a known problem, called in the literature the *bipartite matching problem*. We solve it by using the Hungarian algorithm, which is optimal [Kuhn 1955, Munkres 1957]. We consider only direct, straight-line paths between initial and final positions of the particles, and use the lengths of these paths as the distances required by the Hungarian algorithm.

The matching step is illustrated in Fig.2. The filled green circles in the left panel are the particles in their initial, random positions, and the red crosses denote the goal configuration. The Hungarian algorithm produces the assignments shown on the right panel of the figure. The particles to be moved appear as blue disks, connected to the target positions by the blue line segments, which correspond to the paths that the particles should follow during the pushing operations.

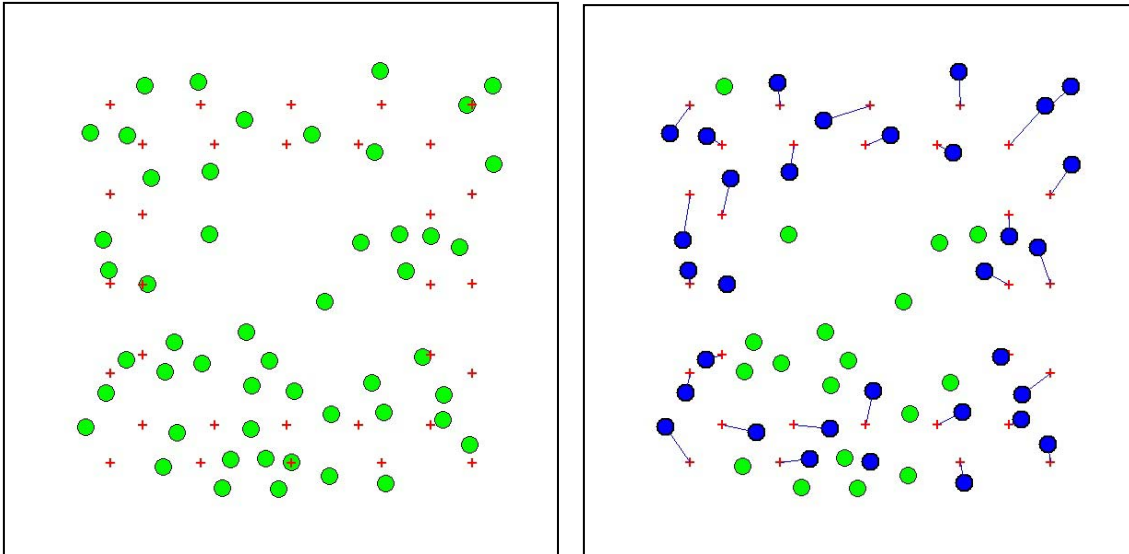


Fig. 2 – Left panel: Initial, random configuration of particles on a substrate surface (green disks), plus goal configuration (crosses). Right panel: Matching obtained by the Hungarian algorithm, which ensures minimal total path length.

3.2 Sequencing

The set of pushing paths calculated in the previous (matching) planner step does not suffice to accomplish our goal. Paths are also needed to move the tip between the end of a push and the start of the next one. This is also a form of bipartite matching, but it is constrained, and does not appear to have been studied in the literature. We solve it by a simple greedy algorithm. We start with an arbitrary target location and find the particle to

be moved (blue) that is closest to that location. We continue the same procedure until all positioning paths have been computed. This algorithm is sub-optimal but in practice it produces good results. We tried to improve the process by using the greedy algorithm as a first step, and then running an optimization algorithm with the results of the greedy computation as initial conditions. We tried simulated annealing, genetic algorithms and ant-colony optimization and obtained only small improvements with significant increases in computation time. Therefore, we decided that the improvements were not worth pursuing in our research.

Fig. 3 shows on the left panel the results of the greedy algorithm for the example of Fig. 2. The red segments between target locations and blue particles are the positioning paths. Combining the set of positioning paths, which is the output of the greedy algorithm, with the set of pushing paths found by the matching algorithm generates a continuous path for the tip, visiting all the target positions and corresponding particles. Executing the sequence of pushing and positioning commands thus obtained produces the goal pattern shown on the right panel of Fig. 3. (See the discussion below for caveats.)

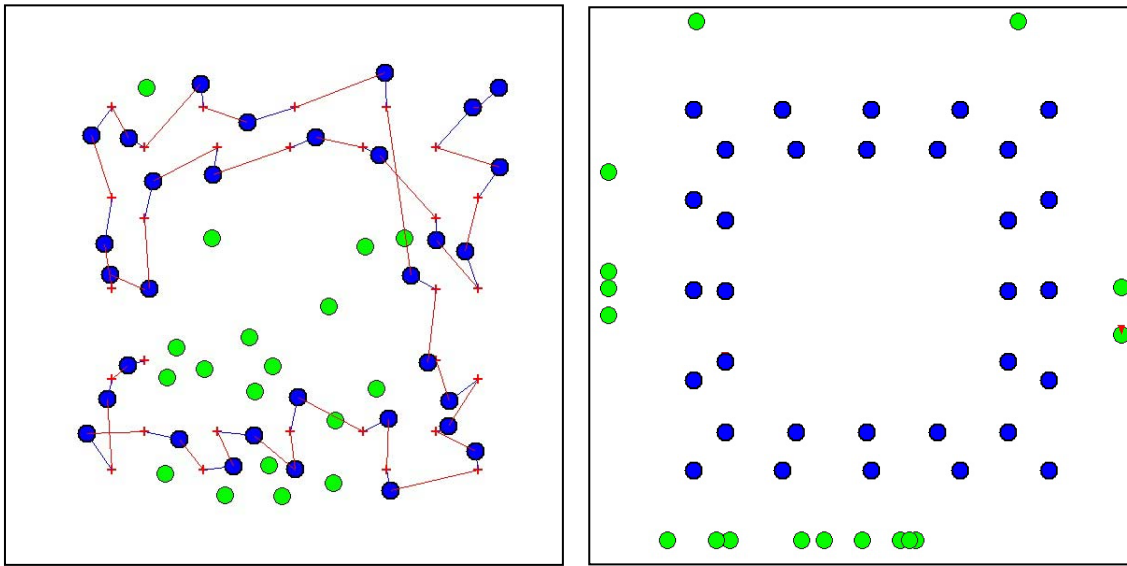


Fig. 3 – Left panel: Positioning motions computed by the greedy algorithm (red) combined with pushing motions produced by the Hungarian algorithm (blue). Right: Result of the simulated execution of the operations shown on the left (plus clearing of extraneous particles—see below).

3.3 Simulated Execution and Collision Handling

The example of Figs. 2 and 3 is very simple. Executing the prescribed sequence of commands does not cause any collisions. But, in a general case, collisions between particles may arise. The planner handles collisions by exploiting the fact that all particles are assumed identical. It simulates the sequence of operations previously computed, at

each step updating the state of the particle arrangement. If a collision is detected, it swaps operations as shown in Fig. 4. Suppose that in the operation sequence the blue particle is to be moved from its initial position 1 to target position 1', and later on the green particle is to be moved from 2 to 2'. The planner can either move the green particle from 2 to 2' first or, alternatively, move the green particle from 2 to 1', followed by moving the blue one from 1 to 2 and then to 2'. Both approaches work, provided that they are applied recursively, because solving one collision problem may generate new ones. The modified path after collision handling is the final output of the planner, and is passed on to the execution system.

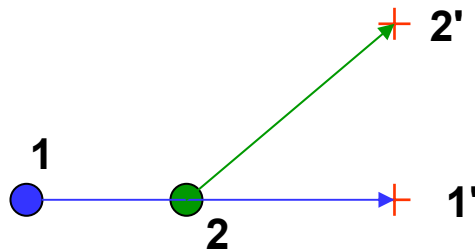


Fig. 4 –Handling collisions between particles.

3.4 Refinements

3.4.1 Extraneous particles

Executing only the sequence of commands shown in Fig. 3, left, does not produce the configuration shown on the right panel of the figure. It places all the required particles on their target locations, but it leaves a dozen or so (green) particles in the middle of the pattern. This is usually undesirable. In this example we simply cleared out these extraneous particles manually, but the planner can do it automatically as follows.

First, a region of interest is specified. This can be done interactively by a user, or automatically by the system, simply by computing a rectangle that is slightly larger than the desired pattern. After running the matching algorithm, several unmatched particles are left inside the region of interest. For each of these, we project its position onto the closest side of the enclosing rectangle. This becomes a new target position, and is added to the goal configuration. When this is done for all the extraneous (i.e., unmatched) particles, we have a new matching problem. We run again the Hungarian algorithm, now with all the initial and added targets and all the particles, extraneous or not, which are within the enclosing rectangle.

Some of the particles that were initially considered extraneous may now become part of the pattern and vice-versa. The Hungarian algorithm ensures that the overall match is optimal. However, moving extraneous particles out to the enclosing rectangle is a heuristic that is reasonable but not optimal, and consequently we can no longer claim optimality for the process of selecting the pattern-building pushing paths.

3.4.2 Static Obstacles

Thus far we have assumed that the only obstacles present in the manipulation area are the particles themselves. However, arbitrary fixed obstacles can be accommodated as follows. First we approximate the obstacles by polygons. Given a set of polygons in the plane and the set of initial and final particle positions, we compute a collision-free path between every particle and every target position. We do this by using the visibility algorithm [Latombe 1991], which is optimal. This computation provides a set of minimal-distance, polygonal-line paths and associated distances. These can be used instead of direct, straight-line distances between the particles and targets in the Hungarian algorithm. The remainder of the planning procedure remains the same. Of course, computing the distances by path-planning techniques is considerably more expensive than just using direct, straight-line Euclidean distances between points in the plane.

4. Execution

The output of the planner is a sequence of primitive positioning and pushing operations, which are straight-line motions, with associated switching of the z feedback off and on for the pushing primitives. Unfortunately, executing these primitive operations is not trivial. As we have seen in Section 2, there are uncertainties associated with the initial positions of the various particles and also with their post-manipulation positions. These uncertainties are so large that the naïve execution of a manipulation plan without the compensations discussed below almost always fails to achieve the desired goal state.

4.1 Drift Compensation

A decade of experimental work at LMR has shown that thermal drift in the x - y plane of the sample is a slow-varying translation, and that the two principal directions can be treated independently. Therefore, the effect of the drift is equivalent to a continuous change of origin for the x - y coordinate system, which can be implemented by adding suitably computed offsets to the x and y coordinates immediately before every motion is executed. Because the drift varies slowly and the motions do not take much time, it is acceptable to keep the offsets constant during the execution of a primitive operation.

We have built a drift compensator based on Kalman filter principles. Details are given in [Mokaberi & Requicha 2006]. Here we summarize the process. The Kalman filter has a simplistic model of the drift, which it uses to predict the drift value at any instant in time. Because the model is not accurate, the error between predicted and real values increases with time. The filter provides standard means for estimating the magnitude of this error, by computing its covariance. When the covariance increases beyond a specified threshold, manipulation operations are suspended and a drift measurement is scheduled. The results of the measurement are combined with the previous estimates according to the usual Kalman filter equations, and the process continues.

The drift is measured by tracking features in a small window, typically 64x64 pixels, which can be specified by a user or automatically computed by the system. The translation that takes place between two instants of time is computed by one of the two following techniques. The first technique involves taking two images of the small tracking window, correlating them and using the largest peak of the correlation to estimate the displacement value. This correlation technique is largely independent of the type of sample being processed. The second method is specific to samples containing approximately spherical nanoparticles. It consists of determining the center of a specified particle at the two instants of time being considered, by executing a few single-line scans. These can be along parallel lines, as discussed in Section 2, or use a “butterfly” search, in which we first scan along the x direction, find the x value that corresponds to the maximum value of the height image of a particle, then scan vertically at that x value to find the y value that corresponds to the maximum of the particle image, then scan horizontally again at that y value, and so forth. A couple of iterations usually provide a good estimate of a particle’s center. In either case, knowledge of the time interval and the corresponding displacement gives us the drift velocity needed for the drift estimation. Note that both techniques need a reasonably good estimate of where the tracked features are, or they will be missed entirely and the process will fail. We ensure that a good initial position estimate is available by using the predicted drift values computed by the Kalman filter to update the tracking window and associated features positions before performing a drift measurement.

Measuring drift requires motion of the tip, and therefore cannot be executed during a primitive manipulation operation. The sequence of positioning and pushing motions must be interrupted for a measurement to take place. Our experience indicates that one can perform manipulation operations for several minutes just on the basis of the drift predicted by the filter. Drift measurements are not needed very frequently. This is an advantage of this approach, because predictions are computed in fractions of a second, whereas measurements require motions of the tip, and typically take tens of seconds.

The videos in Extensions 2 and 3 show the effectiveness of our approach. Both videos depict the evolution in time of a small window within a full-size AFM image. (The large window image is not updated as time evolves.) The objects shown in the videos are 15 nm Au particles on mica coated with poly-L-lysine. The duration of the experiment is 45 minutes, in both cases. In Extension 2 there is no drift compensation. At the end of the 45 minute video, several of the particles have moved out of the tracking window and the drift is large. On the other hand, the video in Extension 3 was recorded with the drift compensator on. Although the compensation is not perfect and a small jitter is observable, the improvement over the non-compensated case is dramatic.

4.2 Creep and Hysteresis Compensation

Drift compensation may suffice for effective nanomanipulation with the latest, top-of-the-line AFMs, which have x-y sensors and feedback loops with sufficiently low noise levels, below ~ 1 nm. None of these, however, are able to compensate for drift, to the best of our knowledge, because the sensors do not measure directly the position of the tip relative to the sample. Furthermore, most of the AFMs in use today either have no x-y

sensors, or their feedback loops are too noisy for successful manipulation of small (~ 10 nm) nanoparticles. For example the Auto Probe (Veeco) instruments we normally use for manipulation at the LMR have x-y sensors, but we prefer to use them in open loop mode when scanning small areas. Because of resolution issues, small-area scans are needed when manipulating small particles. For example, a typical $1\ \mu\text{m} \times 1\ \mu\text{m}$ scan with 256×256 pixels has a pixel size of $\sim 4\ \mu\text{m}$, which is fairly large compared to the 10 or 15 nm diameters of particles such as those shown in the examples in this paper.

We built a feedforward controller that compensates for the joint effects of creep and hysteresis. Details appear in [Mokaberi & Requicha 2008], and here we focus on the basic principles. The fundamental idea is very simple. We construct a model of the creep and hysteresis phenomena that is characterized by the input/output relation $O = f(I)$, where f is a non-linear function. Then we invert this function and implement f^{-1} in software. When we want to move the tip by an amount that would correspond to a voltage X if creep and hysteresis did not exist, we first pass X through the inverse model, and then apply $f^{-1}(X)$ to the piezo motors. This approach is open-loop and relies on having a faithful and *invertible* model.

We construct a joint model of creep and hysteresis by using a Prandtl-Ishlinskii operator, which is guaranteed to be invertible [Mokaberi & Requicha 2008]. Creep is approximated by a linear term plus a superposition of exponentially decaying terms, with different time constants. Hysteresis is modeled by a superposition of *play operators*. A play operator has the input-output characteristic shown in Fig. 5. It is essentially a simple hysteresis loop with a threshold r . The hysteresis model is a linear combination of several of these operators, with different thresholds. The piezo extension is the sum of the values of creep and hysteresis, and can be expressed in terms of a Prandtl-Ishlinskii operator. The combined model depends on several parameters, which can be estimated by analyzing the AFM topography signal for a line scan. The line should span the entire region in which the manipulations will take place and cross several particles, to ensure that the parameters are valid for motions within the selected scanning window.

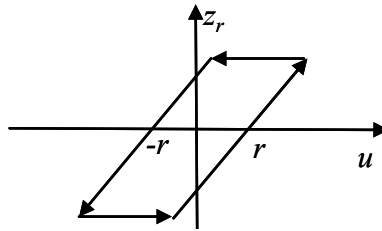


Fig. 5 – Input-output characteristic of a play operator used to model hysteresis.

The videos in Extensions 4 and 5 show that the compensation scheme is effective. Both were obtained by executing a $1\ \mu\text{m}$ motion, followed by scanning back and forth several times along a $400\ \text{nm}$ line that crosses two $15\ \text{nm}$ Au particles. In Extension 4 the creep and hysteresis compensator is off, whereas in Extension 5 it is on. There is a marked

improvement in performance: the traces that correspond to the various scans are nearly coincident when the compensator is on. Quantitative tests show that there is a one order of magnitude improvement by using the compensator.

4.3 Manipulation with Feedback

Combining the two compensators described in the previous subsections produces a software-compensated AFM that is capable of reliably moving between any two specified points. However, this is not sufficient to successfully accomplish manipulation operations because the manipulation itself is unreliable for large displacements, as we have seen in Section 2 and Fig. 1. Therefore, we break down any long pushing trajectory into smaller segments, currently ~ 30 nm long—see Fig. 6. On the top of the figure is the desired task, pushing a particle along a path ~ 60 nm long. We break it into two successive paths of length ~ 30 nm. Before starting the push we measure the center of the particle by using the center-finding procedure described earlier. At the end of the first push we again run the center-finding routine, because the first move may have been slightly inaccurate. Small errors will still let us find the particle, and then the center can be found accurately. (With a large error the procedure would fail, or perhaps find the center of a different particle.) With the new center coordinates we update the trajectory for the second operation, and then execute the push. We emphasize that all this happens automatically, under program control and without user intervention.

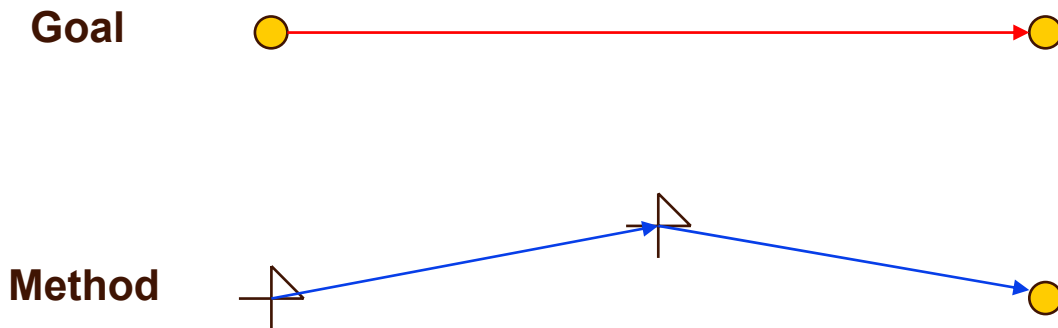


Fig. 6 – Breaking a long path into two short ones, and center-finding before each motion. The first push was somewhat inaccurate, ending out above the correct trajectory. The particle’s measured center coordinates are used to update the second path segment.

The technique just described uses feedback from a previous push to correct the path for the following push. However, this is not true real-time feedback, because the AFM cannot be pushing and center-finding simultaneously. In principle, it is possible to use real-time feedback by exploiting the behavior of some of the signals that can be acquired by the AFM. We have shown that with the LMR protocol the oscillation amplitude used in the tapping mode decreases to zero during a successful push, while at the same time the d.c. (average) deflection of the cantilever increases up to a certain threshold [Baur *et al.* 1998]. Monitoring these two signals would give us useful information to determine if the manipulation is proceeding normally. Unfortunately, to measure amplitude or

deflection and act upon the measured values while the tip is moving requires substantial changes to the inner loops of the controller, and this is usually impossible for a user of a commercial instrument.

5. Software Implementation

The LMR nanomanipulation systems have been implemented, since the lab's beginnings in 1994, on AutoProbe AFMs (Park Scientific Instruments, then Thermomicroscopes, now Veeco). In the mid 1990s, the AutoProbe was the AFM that offered the most convenient programming interface. Even today, its Application Programming Interface (API) is perhaps the most powerful on the market, although it may be somewhat intimidating for users who are not computer scientists. We have developed two software systems for manipulation based on the AutoProbe API: the first is called PCS (Probe Control Software) and the second PyPCS (Python PCS). We describe them briefly in the following subsections.

5.1 PCS

The AutoProbe API is a 16-bit library running under the Windows operating system. Its operation is best explained by a typical example. Suppose that we want to move the tip in a straight line to perform a single-line scan. Assuming that the API has been initialized and the required parameter values have been set, the single-line scan is initiated by calling a specific API routine. These API calls are non-blocking: the API immediately returns control to the calling program, and provides it with a job ID. The API maintains internally its own first-in, first-out job queue. When the request for a single-line scan is received, the job is enqueued. Execution does not take place until the job reaches the beginning of the queue. Then a JOB START message with the job ID is sent through the Windows messaging system. When the job finishes, a JOB END message is generated. It is the calling program's responsibility to process these messages, and retrieve the output data, if any, generated by the procedure call. (The calling program also needs to poll the job queue because sometimes there are race conditions that interfere with the messaging process just described.)

We built the interactive system described in Section 2 by extending the API. Most of the images that have appeared in LMR papers over the last decade have been generated with this interactive system. The execution system discussed in Section 4, including the drift, creep and hysteresis compensators, was also implemented as a 16-bit extension to the API, and used to produce the results presented below, in Section 6. We refer to this system as PCS, for Probe Control Software. The high level planner of Section 3 has not been implemented in PCS and it is not seamlessly integrated with it. It must be run separately and its output provided to PCS.

PCS has served us well but it has major drawbacks: its event-driven, Windows-based, 16-bit architecture makes software development very inconvenient, and forces us to write for the PC platform. These are serious problems, because research software is continually evolving, and flexibility and ease of development are very important attributes of a good

research tool. To facilitate further software development, we embarked on a complete reimplementation, which is described in the next subsection.

5.2 PyPCS

The new implementation is called PyPCS, for Python PCS. It has a client-server architecture, with a server written in C++ for the Windows 16-bit architecture, and a client written in Python. The server must run in the PC that controls the AFM and is built on top of the API. The client can run in any machine that supports Python and is connected to the server by Ethernet. Client-server communication uses standard interprocess communication primitives. Python is a very convenient language for the rapid development of research software. For example, new procedures may be incorporated and tested without recompiling and linking the system.

Fig. 7 is a block diagram of the system. The server is a thin wrapper around the API. Because of the API architecture, briefly described above, the server must listen to the Windows queue, and poll the API internal queue as well. The server also must decode requests from the client, and encode and package data that become available for transmission to the client. All the data going back and forth between client and server are encoded into standard XML (Extensible Markup Language).

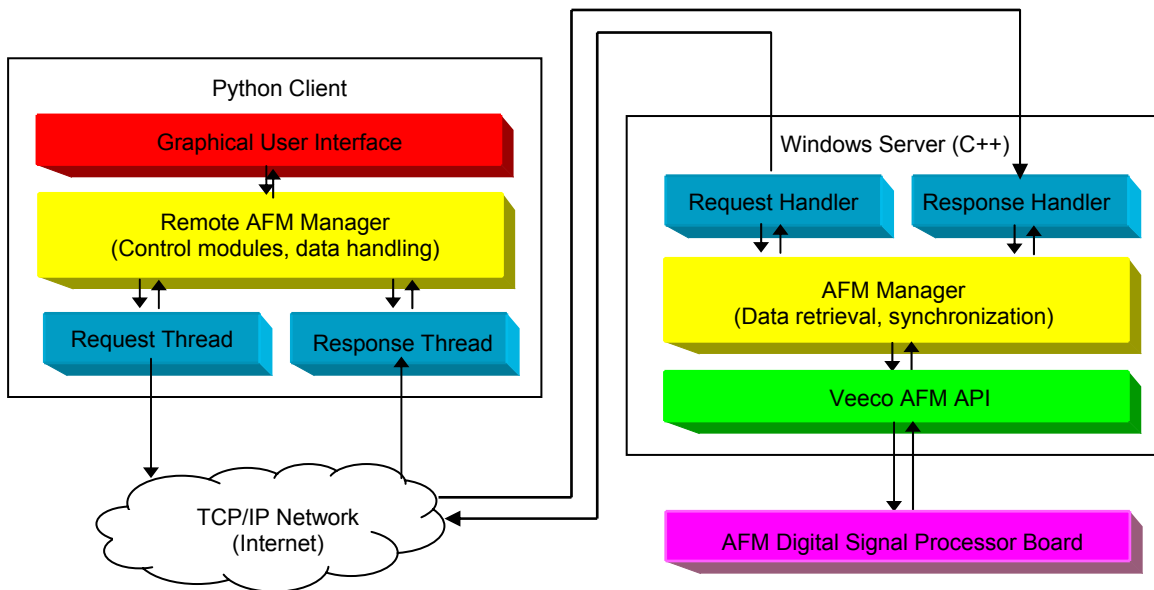


Fig. 7 – Block diagram of the PyPCS system.

The client consists of a GUI (graphical user interface), a manager that implements control module logic, and two threads, one for issuing server requests, the other for processing responses. We have implemented control modules that reproduce and extend the functionality of most of the original PCS. For example, there are modules for executing single-line and rectangular scans, finding particle centers, pushing particles, and so on.

These modules can be composed; for example, the center-finding module uses the single-line scan module.

A brief (and slightly simplified) example will serve to illustrate the basic functioning of PyPCS. Let us consider again a single-line scan. The process begins on the client side with a call to a single-line scan module. This control module, in turn, invokes the corresponding client-side remote procedure call stub for the server interface. The thread in which the module is running blocks and waits for the data requested from the server. The stub packages the request for transmission to the server using a protocol built on top of XML-RPC [Winer 2003].

The server receives the request, unpackages it, and calls the appropriate API function. Then it listens for messages on the Windows message queue and also polls the API. Eventually the requested job is executed and the data are acquired by the server, which packages it using XML-RPC and sends the information back to the client. The data are decoded on the client side and made available to the single-line scan module, which unblocks, terminates execution and returns the scan data.

This example shows that PyPCS supports a simple, imperative programming style, and hides the complexities of the API event-driven 16 bit architecture, as well as the DSP (digital signal processor) details.

PyPCS reproduces the functionality of the interactive part of the original PCS and is much more convenient and extensible. It also is integrated with the high-level planner, which is written primarily in Python.

However, we have found serious problems in implementing in PyPCS the compensators described in Section 4. There are timing issues that we have not been able to resolve, and our current view is that the compensators need to be implemented on the server side. We have not yet done this.

6. Results

Fig. 8 shows in the left panel an initial, random dispersion of Au nanoparticles with diameters of 15 nm, on a mica surface coated with poly-L-lysine. The planner-computed paths are also shown. On the right panel of the figure is the nanoparticle pattern, a uniform array, that results from executing in PCS the paths shown on the left.

The video in Extension 6 depicts a construction task for a triangular structure, starting from a random configuration of 15 nm Au particles. First, a small window is selected for drift measurements. The Kalman filter runs for a few minutes until the covariance of the error is small. The pushing operations are then started. The video shows the paths taken by the tip superposed on the initial particle configuration. Several center-finding “butterfly searches” are evident. Note that we do not have real-time images of the results of the pushing operations, because this would require interrupting the manipulation for a

significant amount of time. After all the manipulation is done, we image the result, showing the desired triangular pattern.

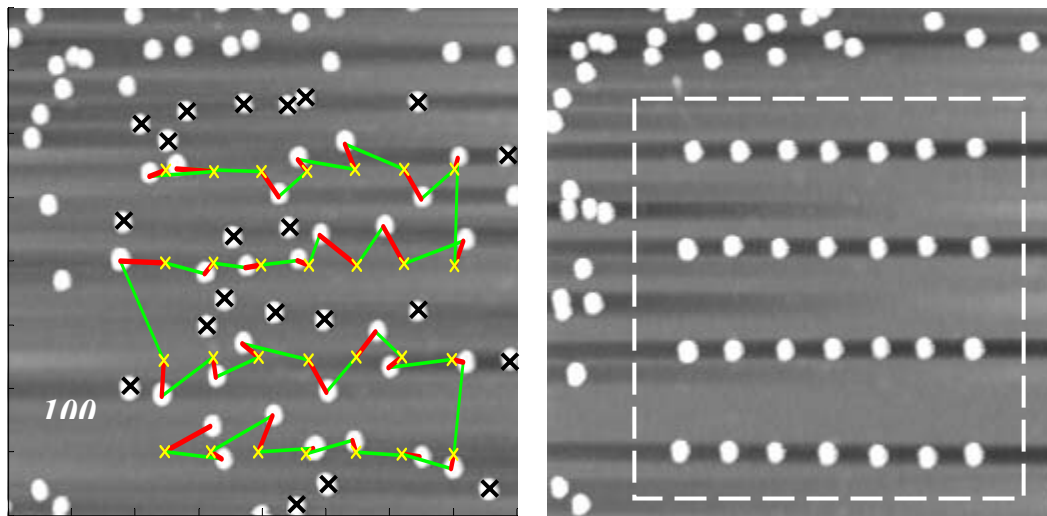


Fig. 8 – Left: initial configuration of 15 nm Au particles on mica coated with poly-L-lysine, and pushing and positioning paths computed by the planner. The crossed particles are extraneous and were removed by interactive manipulation. Right: the resulting array.

7. Conclusion and Outlook

The automated nanomanipulation system described in this paper is capable of building structures such as those shown in Fig. 8 and Extension 6 in a matter of minutes, whereas the interactive systems of the past would require several hours of a skilled operator to construct similar, but usually less accurate, patterns. Structures of unprecedented complexity, made of nanoparticles with sizes on the order of a few nanometers, can now be conveniently built by AFM nanomanipulation. The full impact of this new capability is yet to be explored.

We showed that it is possible to compensate in software for the nonlinearities of the AFM drives. This is important not only for manipulation but also for AFM-based nanolithography, if accuracies on the order of the nanometer are desired.

Programmability of AFM software systems is essential for the work reported here, and, more generally, for many innovative applications of AFM. Yet, commercial systems tend to have very limited programmability, often aimed at given end-users, who are not computer scientists, a few useful capabilities. We contend that the ability to program and modify just about any piece of the AFM control software is highly desirable, and that these capabilities should be directed towards teams such as ours that include computer scientists, control engineers, and the natural scientists who are the main end-users. Current AFMs are very sophisticated pieces of hardware but tend to have relatively primitive control schemes and software. Building a generic, highly programmable and

extensible AFM controller would be a very worthwhile effort that would enable research on novel control and software systems to support new applications.

Acknowledgements

The research reported here was supported in part by the NSF under Grants EIA-98-71775, IIS-99-87977, EIA-01-21141, and DMI-02-09678. Earlier versions of portions of this paper were presented at the Int'l Advanced Robotics Programme (IARP) Workshop on Micro & Nano Robotics, Paris, France, October 23-24, 2006, and at the IEEE Int'l Conf. on Robotics & Automation (ICRA '07), Rome, Italy, April 10-14, 2007.

Jaehon Yun built the planner, Babak Mokaberi designed and implemented the compensators, and Dan Arbuckle architected the automated system described in this paper. The original PCS was built in the late 1990s by Cenk Gazen and Nick Montoya, and the original PyPCS was developed by Jonathan Kelly. All of the above named contributors were graduate students in the LMR at the time. We would also like to thank our collaborators from Physics and Chemistry, too many to name here, without whom we could not have built the systems described in this paper.

References

[Baur et al. 1998] C. Baur, A. Bugacov, B. E. Koel, A. Madhukar, N. Montoya, T. R. Ramachandran, A. A. G. Requicha, R. Resch, and P. Will, "Nanoparticle manipulation by mechanical pushing: underlying phenomena and real-time monitoring", *Nanotechnology*, Vol. 9, No. 4, pp. 360-364, December 1998.

[Kuhn 1955] H. W. Kuhn, "The Hungarian Method for the assignment problem", *Naval Research Logistic Quarterly*, Vol. 2, pp. 83-97, 1955.

[Latombe 1991] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[Mokaberi & Requicha 2006] B. Mokaberi and A. A. G. Requicha, "Drift compensation for automatic nanomanipulation with scanning probe microscopes", *IEEE Trans. on Automation Science & Engineering*, Vol. 3, No. 3, pp. 199-207, July 2006.

[Mokaberi & Requicha 2008] B. Mokaberi and A. A. G. Requicha, "Compensation of scanner creep and hysteresis for AFM nanomanipulation", *IEEE Trans. on Automation Science & Engineering*, in press (on line at IEEE Xplore).

[Munkres 1957] J. Munkres, "Algorithms for the assignment and transportation problems", *J. Soc. Indust. Appl. Math.*, Vol. 5, pp. 32-38, March 1957.

[Requicha 2003] A. A. G. Requicha, “Nanorobots, NEMS and nanoassembly”, *Proc. IEEE*, Vol. 91, No. 11, pp. 1922-1933, November 2003.

[Requicha 2008] A. A. G. Requicha, “Nanomanipulation with the Atomic Force Microscope”, in R. Waser, Ed., *Nanotechnology for Information Processing*. Weinheim, Germany: Wiley-VCH, 2008, Chapter 8.

[Winer 2003] D. Winer, “XML-RPC specification”, <http://www.xmlrpc.com/spec>, June 2003.

Appendix – Index to Multimedia Extensions

The multimedia Extensions to this article can be found online by following the hyperlinks from www.ijrr.org.

Extension	Media Type	Description
1	Video	Interactive pushing with PCS
2	Video	Drift in a small window without compensation
3	Video	Drift-compensated image of small window
4	Video	Back and forth single-line scans without creep and hysteresis compensation
5	Video	Creep and hysteresis compensated single-line scans
6	Video	Automatic construction of a triangular nanostructure