In the Proceedings of Intelligent Systems and Control ~ISC 2007~, November 19-21, 2007, Cambridge, MA, USA

MOTION PLANNING AND ASSEMBLY FOR MICROASSEMBLY WORKSTATION

Asanterabi Malima and Asif Sabanovic
Faculty of Engineering and Natural Sciences
Sabancı University, Tuzla, İstanbul, Turkey
malima@su.sabanciuniv.edu and asif@sabanciuniv.edu

ABSTRACT
In general, mechatronics systems have no standard operating system that could be used for planning and control when these complex devices are running. The goal of this paper is to formulate a work platform that can be used as a method for obtaining precision in the manipulation of micro-entities using micro-scale manipulation tools for microsystem applications. This paper provide groundwork for motion planning and assembly of the Micro-Assembly Workstation (MAW) manipulation system. To demonstrate the feasibility of the idea, the paper implements some of the motion planning algorithms; it investigates the performance of the conventional Euclidean distance algorithm (EDA), artificial potential fields' algorithm, and A* algorithm when implemented on a virtual space.

KEY WORDS
Motion planning, microassembly, path planning algorithms, object oriented programming

1. Introduction

Our goal is to formulate a work platform that can be used as a method for obtaining precision in the manipulation of micro/nano-entities using micro/nano-scale manipulation tools when such platform is installed in micromanipulation system. In attacking the problem we elaborate the functionality of such platform using several motion planning algorithms while performing controlled manipulations. What motivates us for this work is to reduce human intervention when conducting semi-autonomous manipulation in the MAW. To achieve that we need to specify tasks in a high level language and have the manipulators and stages automatically convert these specifications into a set of low level primitives to accomplish tasks.

In literature, motion planning and control of single and multi-agent robotic systems is a very challenging problem that received a lot of attention in recent years [1]. Challenges in motion planning arise from development of a computationally efficient framework accommodating both the complexity of the environment and the manipulation system control along with the communication constraints, while allowing for a 'rich' specification language [1]. In [3], an attempt to discuss planning in micro/nano mechatronics technology such as, precision positioning, micro/nano actuators and microgrippers together with use of macro mechatronics technology in mechanism for intelligent robot control and advanced-user interface can be seen. However, one of the approaches taken in this work is to translate the motion planning problem into a 'least cost path' graph problem with an associated cost function for trajectories on the graph and a simple progressive scheme is needed for very large graphs giving approximate solutions in reasonable time and memory space [4]. In [4], the motion planning is associated with pathfinding algorithm, so as to determine first, if it is possible to find a path from the start point to the goal on the map whereby maps may contain impassable barriers, objective is to be able to determine the optimal path. There are many factors that are evaluated to determine the optimal path (defined as the path with the least cost) including the speed in which a path will be traversed. Noticeably, there are many different approaches for defining a path ranging from simple (walk forward until you hit something) to the complex (path finding algorithms with heuristics) [6].

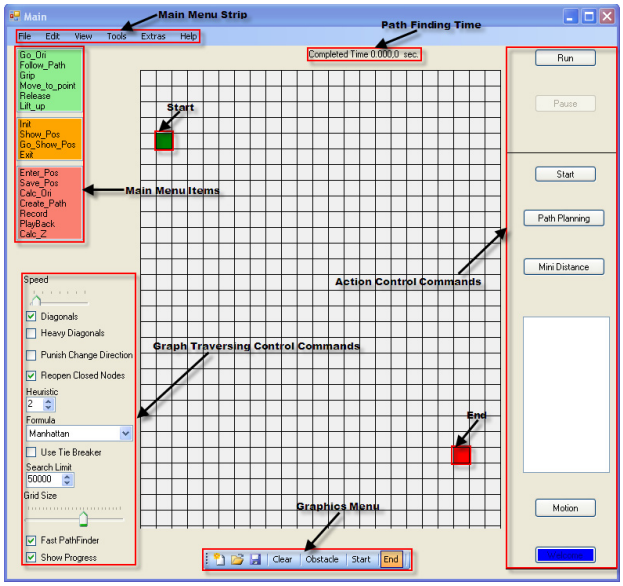

Figure 1: Window application used to control operations of the MAW

The aim of this paper is to present a command language for our MAW, an idea which was previously introduced in [7] in order to reduce human intervention in the semi-automated manipulation process of MAW [8]. Along with this feature we present common and simple algorithms that are used to demonstrate the functionality

of our application. To achieve this automated precise manipulation we formulate a windows application tool in object oriented programming enviroment which can be used to easily interfaced with other environment such as signal processing board and image processing tools. Such kind of window application (see Figure 1) will provide GUI interface which would enable the operator to change the relative position and relation of entities through direct or indirect human operator control [9]. Given list set of commands on windows application, with motion planning algorithm, we explore the ability of our system to plan path around obstacles or to make choice of the best sequence of arrangements in moving particles to their destinations without any collision with obstacles. An interesting feature of our work is to plan motion around obstacles on a map purposely to demonstrate functionality of command language in performing autonomous manipulation of objects in microsystems.

The section 2 deals with the software for motion planning and control of the MAW. The section 3 contains a theoretical discussion of the motion planning algorithm used for micromanipulation. In subsection 3.1, the discussion about the proximity (Euclidean distance) based algorithm for manipulation of obstacles followed by the subsection 3.2, which explores the features of the artificial potential field approach, while the subsection 3.3 covers the concepts of the graph traversing algorithm A* (A star). Then, section 4 covers the results obtained followed by conclusion and discussion on further work to be conducted in this research project.

2. Software for Object Manipulation

As discussed earlier, our motivation for this work is to create a language in an object oriented fashion in which a user would give some commands and the system would respond accordingly to execute the tasks. In attacking the problem, the motion plan is obtained through model checking, and results in the form of graphical representation of the provided workspace. A brief discussion on the structure and architecture of the software is covered in this section. The following section will also cover discussion of how the software architecture results to computational complexity. The manipulation tasks in virtual space are specified by identifying a target object to be moved and its new location (see Figure 2).

2.1 Software Architecture

We prepared the list set of commands presented in the *divide-and-conquer approach* for the identification of control modes and their combination can lead to major task in the manipulation process with emphasis on precision, accuracy, reliability and repeatability. To achieve this we prepared classes for the whole system. Then, all the system is divided into several categories with particles, stages, microscope and manipulation tools as **objects** which are represented by the defined classes. e.g. For the real time experiment we have polysterene ball images represented with the following **properties**: circular objects in 2D, with a given radius r ,and x and y coordinates of the center. Collection of these mentioned properties can be used to represent **methods** such as rotate, push, record coordinates of particular particle**.**

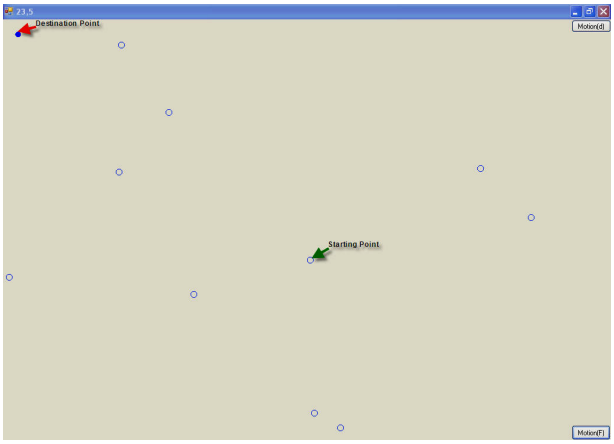

Figure 2: Particles randomly distributed on the virtual space

2.2 Structure of the Software

Based on the structure presented in [7], motion planning window application software can be divided into three categories which are *Movement Commands*: Motion commands such as “Follow Path”, “Move to Point”, “Lift Up”, “Rotate(change orientation)” and “Return to Origin” are categorized in this part; *Action Commands*: Commands to “Initialize Manipulation”, “Show Position of the Moving Part” and “Exit Manipulation” actions are presented; and Definition *Commands*: List of basic definition manipulation of particles such as “Enter Position”, “Save Position”, “Select Motion Planning Algorithm”, “Create Path”, “Save Movie”, “Play Movie”, etc. In order to prepare the main window application some features were taken from [10], these include:

- *Grid Size*: This parameter just affects the front-end. It can change the grid size, where reducing the grid size gives a chance to create a bigger test but will take longer to render.
- *Fast Pathfinder*: When unchecked, the implementation uses the algorithm as it appears in Path Finder [10]. When checked, it uses path finder implementation which requires more memory. However, using fast Pathfinder makes the search about 300 to 1500 times faster depending on the map complexity.
- *Speed*: This is the rendering speed of the Pathfinder; reducing speed permits detailed examination of how the algorithm opens and closes the nodes in real-time while the Pathfinder operates.
- *Diagonals*: Is set to allow the A* algorithm to process path searching in 8 directions instead of 4; including the diagonals of the grid.
- *Reopen Closed Nodes*: Is the command that allows A* algorithm to reopen nodes that were already closed when the cost is less than the previous value. If reopen nodes is allowed it will produce a better and smoother path, but it will take more time.

- *Formula*: Is the equation used to calculate the heuristic. Different formulas will give different results: some will be faster, others slower and the end may vary.
- *Punish Change Direction*: Is the command that allows every time the A* algorithm path finder changes direction the cost decreases. The end result is that if the path is found it will be comparatively smooth without too many direction changes, thus looking more natural. The downside is that it will take more time because it must research for extra nodes.
- *Show Progress*: This permits observation of the algorithm as it operates in real-time. If this box is checked, the completion time will be the calculation time plus the rendering time.
- *Tie Breaker*: In A* path planning algorithm sometimes it encounters a phenomena in which there are many possible choices for the same cost and destination. The tie breaker setting tells the algorithm that when it has multiple choices to research, instead it should keep going. As it goes, changing costs can be used in a second formula to determine the "best guess" to follow.
- *Completed Time*: Is the time the algorithm takes to calculate the path from the start to end point. To know the true value, uncheck "Show Progress."
- *Run/Continue and Pause* Are action control command buttons use to run and control the A* motion planning algorithm. The Pause command is used to temporary stop the graph traversing process which the search for the path continues.

All these functions are featured in the window application main form as seen in Figure 1 and are used in the operation of the software. Also seen in Figure 1 is the "Action Control Commands" which contains the "*Path Planning*"; command used to initialize the virtual workspace form (see Figure 2), so that EDA and potential field oriented motion planning algorithm can be used to determine path for manipulation of randomly distributed particles.

2.3 Operation of the Software

Manipulation Process: "Initialize Manipulation – Action command" initializes the manipulation process, in which the manipulators move to the origin(home) position which is known, "Select Algorithm – Definition command" enables user to select the appropriate algorithm(to be discussed in the next section) to used in the particular task. "Save Movie" is the command used to save the manipulation process as avi.file so that it can be replayed using "Play Movie – Definition command" later for further analysis. "Return to Origin – Movement command" is the command for return all stages and manipulators to the start (system initializing position) and "Exit Manipulation – Action command" is used for stoping the manipulation and switching-off the application. By default, program can also be set in such a way that once the object has been placed at the target(destination) location, the manipulator can return to its rest position (system home position). Also, in the execution of the commands we assign hierrachies among the commands to ensure that priorities are given to a set list of commands e.g. Action commands. Our project challenges involves writing the codes for the commands which run multiple systems at once. This feature, however may lead to several outcomes such as delay in debugging since handling all the data coming from multiple sensors(camera, motors and manipulators) and sending the appropriate commands to the motors and the manipulating tools. Implementation of such commands can be observed in the motion planning algorithms in the next section.

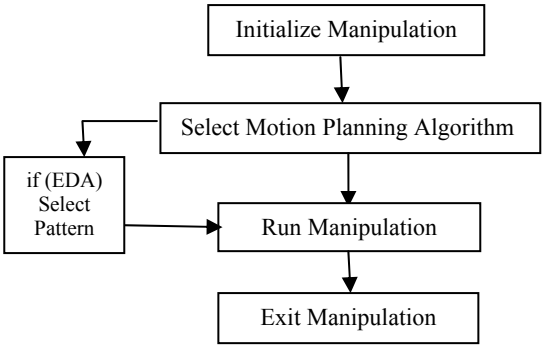

Figure 3: The manipulation procedure using the software

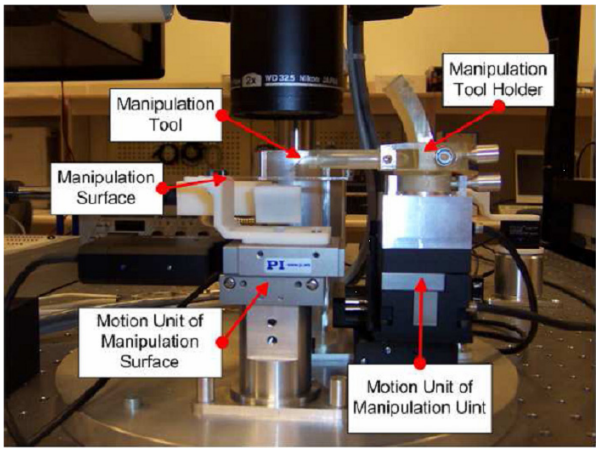

Figure 4: MAW setup with the manipulation surface as our working environment

The window application commands can be put into two categories, the first one involves commands to perform single operations (tasks), and the second one includes collection of systematic list of commands given when system is required to perform certain action. The manner in which the motion planning algorithms is set to perform manipulation of micro-particles fits into the second category of the window application commands.

Figure 4 shows the MAW setup with the standard microscope slide manipulation surface as its workspace for manipulation activities. In Figure 5 example of semi-autonomous manipulation experiment in MAW is demonstrated. Note that in semi-autonomous manipulation, the system initialization includes the

selection of the particle and the desired destination point. Later, a line connecting the center point of the selected particle and the destination particle is constructed. Then, step motion value for pushing the particle and the selected point are calculated. Finally, the tip of the probe is moved in steps towards the destination point in order to achieve semi-automated manipulation. The steps are repeated until the center of the selected particle coincides with the desired point. The most feasible trajectory for a particle to its target position by pushing is simply a line denoting the closest path to the destination. In that context, the operator should choose the suitable part to be pushed and the destination point considering the issue that the semi-automated assembly procedure does not include motion planning, so there exists nothing as an obstacle between the particle and the target point.

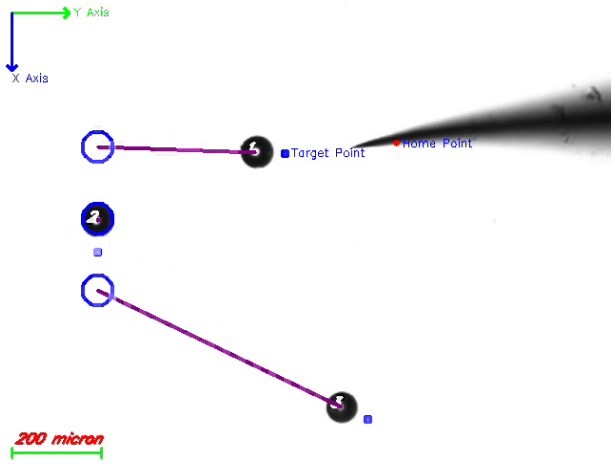

Figure 5: MAW semi-autonomous manipulation example [8]

3. Motion Planning Algorithms

Consider a workspace with randomly distributed micro parts, our objective is to use motion planning intelligent algorithms to formulate patterns of particles in defined locations on the workspace. In motion planning procedure a good understanding of the physical feature of our manipulation system is very crucial before the user issues commands. Then, after all constraints have been put into consideration the application of this technology in assembly of micro components and structures into microsystems can be fulfilled. Common application of this planned manipulation is on construction of useful 2D microstructure, e.g in fabricating mold templates in micro/nanoprinting [2].

With few exceptions, most of the works in this area focus on either the complexity of the environment or manipulator dynamics (while assuming the environmental features are trivial). Previous study show communication architectures in multi-actuator systems focusing on proving that certain local interactions give rise to interesting global behaviors. However, the inverse problem of generating local rules from non-trivial high level specifications of the group is still not understood. In most of the existing works, the motion planning problem is simply specified as "go from A to B". It has been discussed by several authors [5] that this kind of observation is either too explicit, or simply does not capture the nature of the task, which might require logical (e.g., "visit either A or B") and/or temporal operators ("reach A and then B infinitely often").

In this paper we conduct our analysis for three different types of motion algorithm:

- Conventional Euclidean distance algorithm, in which the particle with the nearest proximity to the destination is the one which is pushed by the manipulator.
- Potential Field algorithm, whereby our virtual space is considered to be covered by artificial potential fields, however the algorithm works as to only take into consideration the field potential originating from the particle itself, destination and obstacle particles close and along the path between origin and destination of the moving piece [11].
- The A* algorithm is implemented on our application. It should be noted that, what sets A* apart from best-first search and early algorithms is that it calculates the path with a minimum cost, and also takes the distance already traveled into account. This makes A* *complete* and *optimal*, i.e., A* will always find the shortest route if any exists and if heuristic was chosen correctly. However, it is not guaranteed to perform better than simpler search algorithms.

In the following subsections we will give detailed explanations of how algorithms were applied in the virtual space of our window application.

3.1 Euclidean Distance Algorithm

This algorithm defines the geometric representations of all particles in the workspace, and that of the manipulator as well. The method allows planning algorithms to determine whether particle being moved by the manipulator is in collision with other particles or with obstacles. The idea is that, the particles that are closest to the destination are being pushed towards their proposed destinations.

Given an initial coordinate frame $F := (OXY)$ on our workspace with origin O and axes X,Y, the position of manipulator is determined by its coordinates (x_i,y_i). Then, if the position of the manipulator at any time (t) is $(x_i(t), y_i(t))$ and let *q(t)* be the configuration variables, then

$$x_i(t) = x(t) + d_i \cos(\theta(t) + \alpha_i), \quad (1)$$

$$y_i(t) = y(t) + d_i \sin(\theta(t) + \alpha_i). \quad (2)$$

where, d_i:=is the distance between original position of the particles and their destinations and α_i is the angle formed by the segments d_i and the positive horizontal axis.

Let us introduce the distance function:

$$L(q(t), \dot{q}(t)) = \left[(\dot{x}(t) - d_i \sin(\theta(t) + \alpha_i)\dot{\theta}(t))^2 + (\dot{y}(t) + d_i \cos(\theta(t) + \alpha_i)\dot{\theta}(t))^2\right]^{1/2} \quad (3)$$

Since for 2D manipulations the cost function is the distance between two points, our goal would be to minimize the function in equation (3), so that the

manipulation is performed for the particles which are closest to the destination.

$$Path = \arg \min L(q(t), \dot{q}(t)) \quad (4)$$

In virtual space we consider the random distributed particles with the same physical properties. Our objective is to move the particles to their defined destinations without any collisions between them during the manipulation operation. Since the particles are spherical, in 2D the particles have the same radii; therefore there is no question whether the manipulated particle will fit well into their assembled locations. Assuming this situation is true the manipulations begins with calculating the closest particle to the destination point in consideration. Then, a line connecting the defined destination location and the closest particle is drawn. Afterwards, the line is distributed into segments depending on the structure of movement in the manipulation. Finally, the manipulation process starts by slowly pushing the particle towards the destination following the straight line. The Figure 6 below demonstrates how the process is implemented in virtual space with particles of radius 10 units. Similarly Figure 5 showed how the same algorithm can be applied in the MAW setup using semi-autonomous manipulation features already available in the setup.

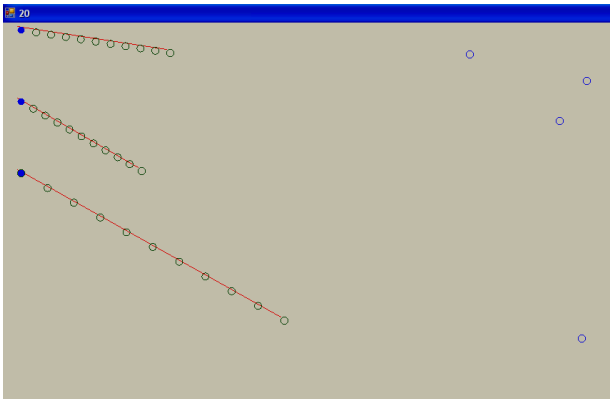

Figure 6: Motion of particle with the proximity based algorithm operating

3.2 Artificial Potential Fields Algorithm

Manipulation use EDA considers microparticles having the same physical properties (mass, area and volume), therefore the necessary force required to move the particles is the same for all the particles in the workspace. However, in real situation the dimensions of the particle located on the workspace are of different magnitudes which prompts for the selection of which particle to move and when to move. In consideration of the above mentioned scenario the necessity of the system to have the motion planning algorithm which has intelligence to allow manipulation of particle around obstacles as it moves from their original locations to their defined destinations seems to be inevitable. This scenario prompts the introduction of artificial potential fields' algorithm into our window application. In Artificial Potential Fields method, an obstacle applies repulsive forces on the manipulator, simultenously the goal applies an attractive force to attract the manipulator and the particle being pushed towards its direction. Eventually the manipulator is forced to take the direction of the resultant force field.

One of the challenges with artificial potential field is the problem of local minima [12]; we had to find a way to get out of local minima in our solution or have no local minima at all. To accomplish this we had to modify the way in which we build our potential field. If the obstacle force experienced by the moving particle is as shown in Eqn. 5 below:

$$\vec{F}_{obs} = -O \cdot \sum_{i=1}^{n} \frac{1}{d_i^2} \cdot \hat{r}_i \quad (5)$$

where O is a constant scaling factor, n is the number of obstacles, d_i is the distance between obstacle i and the manipulator, $\hat{r}_i$ is the direction vector from moving particle to representative point of obstacle. Note that, in equation (5) O is divided by d_i^2, therefore, the obstacle force increases as the moving particle gets closer to the obstacle(as d decreases). The obstacle force comes into consideration once the obstacle reaches the perimeter of the dotted circle of Figure 7, otherwise the obstacle force is negligible. The attaraction force between the moving piece and the goal is:

$$\vec{F}_{goal} = G \cdot d^2 \cdot \hat{r} \quad (6)$$

where G is similarly a scaling factor, d is the distance from moving particle to the goal, and $\hat{r}$ is the direction vector from manipulator and moving particle to the goal.

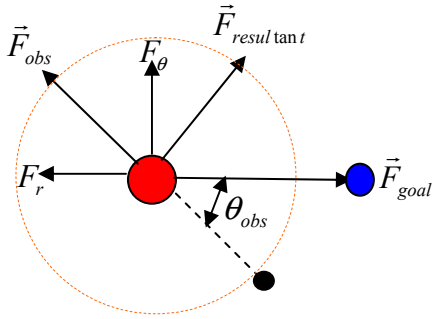

Figure 7: Motion of particle with the artificial potential field algorithm

Using the outputs of the distance sensors, the net repulsive force is calculated and decomposed to its components, one along the direction of motion of the manipulator and one perpendicular to it. If θ_{obs} is the angle between the direction of motion of the manipulator and the obstacle force, then

$$F_r = \left|\vec{F}_{obs}\right| \cdot \cos(\theta_{obs})$$
$$F_\theta = \left|\vec{F}_{obs}\right| \cdot \sin(\theta_{obs}) \quad (7)$$

For safe motion of the moving piece being pushed, the manipulator should try to keep the force component along its direction of motion, F_r, minimum or ideally zero. This can be achieved by changing the orientation of the manipulator, since the force components are dependent on the orientation. To this end, a controller can

be used for the optimization. The rate of change of the force components with respect to the obstacle angle is,

$$\dot{F}_r = -\left|\vec{F}_{obs}\right| \cdot \dot{\theta}_{obs} \cdot \sin(\theta_{obs}) \qquad (8)$$
$$\dot{F}_\theta = \left|\vec{F}_{obs}\right| \cdot \dot{\theta}_{obs} \cdot \cos(\theta_{obs})$$

Then, the controls to drive the PZT 3-axis manipulator which hold the manipulator are selected as $u_r = \dot{F}_r$ and $u_\theta = \dot{F}_\theta$.

The techniques of using potential fields can be seen an interesting alternative for the A* algorithm that seems to be more popular in the current state of the art. The potential fields require some serious calculations which are computationally expensive [11]; however, with modification in which computations for the potential fields are done only in the vicinity of the moving particle, the resulting operation becomes less computationally expensive.

For the modification of the artificial potential field algorithm; first after selecting the particle to be manipulated, a line connecting the defined destination position and the particle to be manipulated is drawn. Then the particles close to the line joining the start and the end position of the manipulation are treated as obstacles as shown in the Figure 8 below. For software implementation in virtual space the moving particle only takes into consideration its distances from the particles along its path and its desired destination.

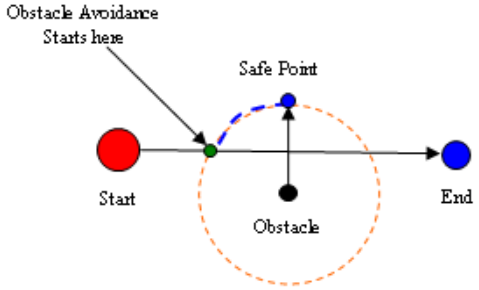

Figure 8: Obstacle Avoidance path of the artificial potential field algorithm

3.3 Graph Traversing A* Algorithm

Finally, we implement the A* algorithm, a popular path finding algorithm used to find a shortest path. A*'s functionality resembles that of Best Fit Searching (BFS) in that it can use a heuristic to guide itself. A* algorithm incrementally builds all routes leading from the starting point until it finds one that reaches the goal. But, like all informed search algorithms, it only builds routes that appear to lead towards the goal. In order to determine which routes will likely lead to the goal, A* algorithm employs a heuristic estimation of the distance from a given point to a specified goal. In the standard terminology used while talking about A*, the following equation holds:

$$f(n) = g(n) + h(n) \qquad (9)$$

where: - n is the node (vertex) of the current location of the moving particle and manipulator
- g (n) represents the cost of the path from the starting point to any vertex n
- h (n) represents the heuristic estimated cost from vertex n to the goal

In this case since the objective is to find the route, h (n) may be the straight-line distance (see figure 15) or else depending on the heuristic. Formulation of the A* algorithm heuristic function h[v] is generally the 'distance from u to v' times 'smallest terrain cost' is the best estimate we can do for h[v] when an underestimate must be guaranteed. Further details on theory behind the A* algorithm can be seen in [6].

In the following section, we provide results when we studied the feasibility for constructing 2D microparticles/ microstructure autonomously using image processing and motion planning algorithms.

4. Verification for Assembling

The results will show verification for assembling performed on the virtual space, to be followed by the results obtained from the experiments on the MAW setup in future.

4.1 EDA Results: Shows the snapshots in assembling letter "I"

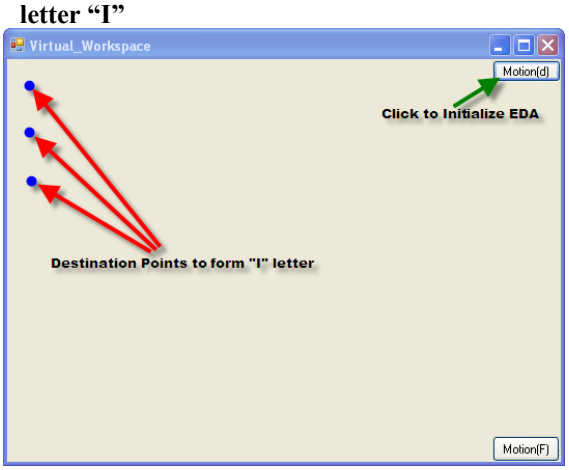

Figure 9: Initializing Manipulation Path

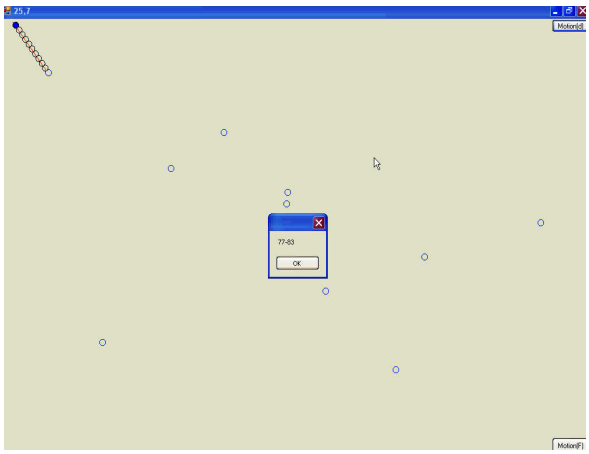

Figure 10: First Particle's Path

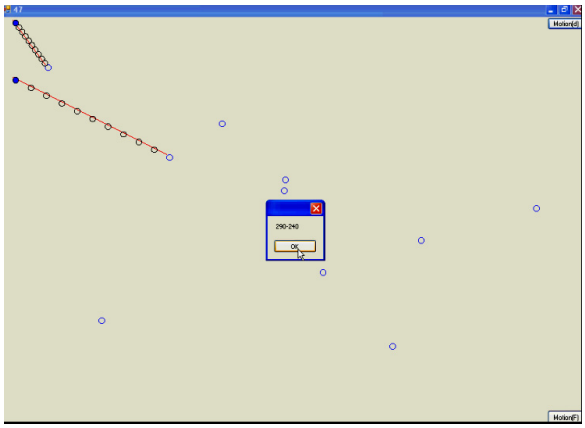

Figure 11: Second Particle's Path

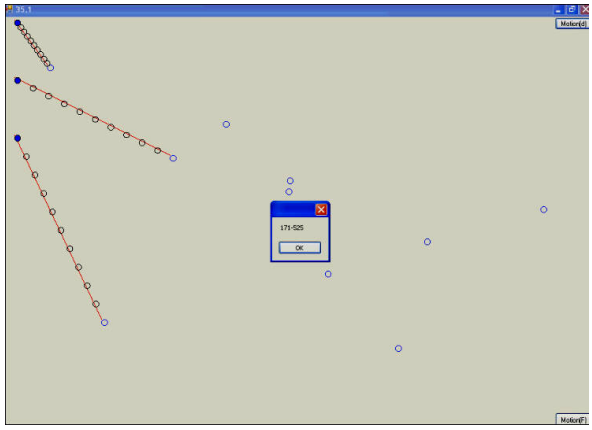

Figure 12: Third Particle's Path

4.2 Artificial Potential Field Algorithm (PFA) Results:

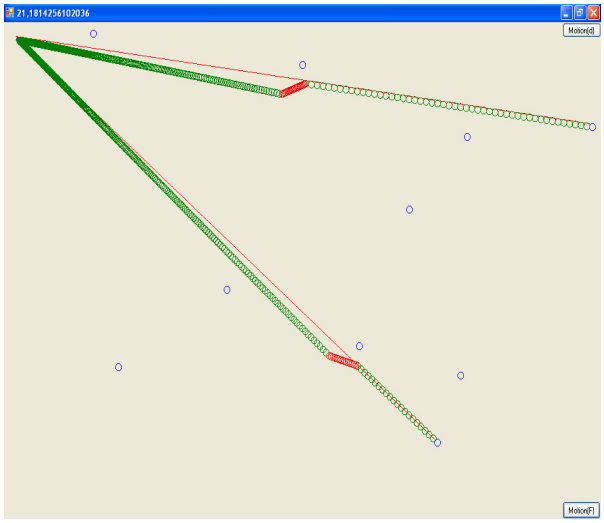

Figure 13: First example of manipulation using PFA

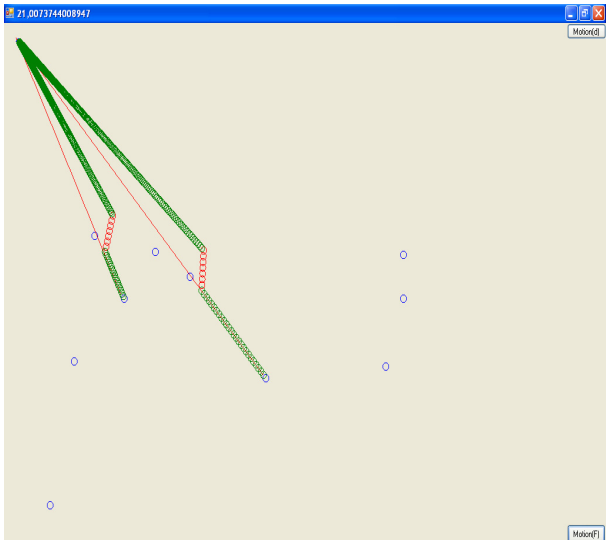

Figure 14: Second example of manipulation using PFA

4.3 A* Results

A commonly used admissible heuristic in motion planning is the straight line distance to the goal, although other heuristics are possible (see Figure 15).

For motion without obstacle:

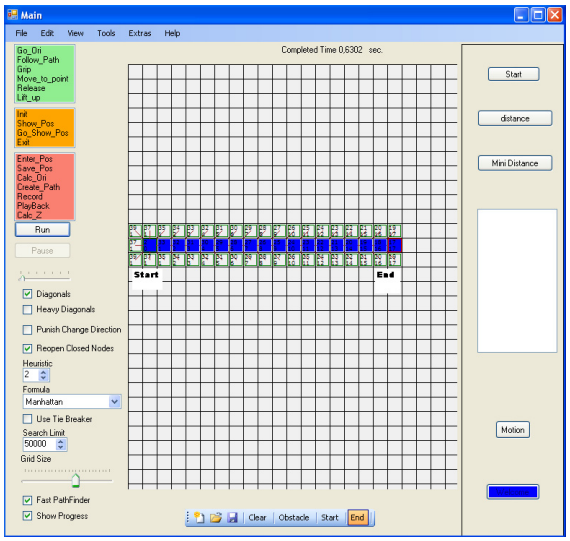

Figure 15: Motion of particle using A* algorithm (no obstacles)

We are currently implementing the path finder system to help with the experiments on our MAW, based on the motion planning algorithms discussed in this paper. Our system profile (windows application) has interactive pull-down menus and 2D map to show the user the planned path and other options to go with the setup as discussed in the previous section. The simulation results of our system demonstrate its potential for interaction, and manipulation of micro systems.

For motion with obstacle on the way:

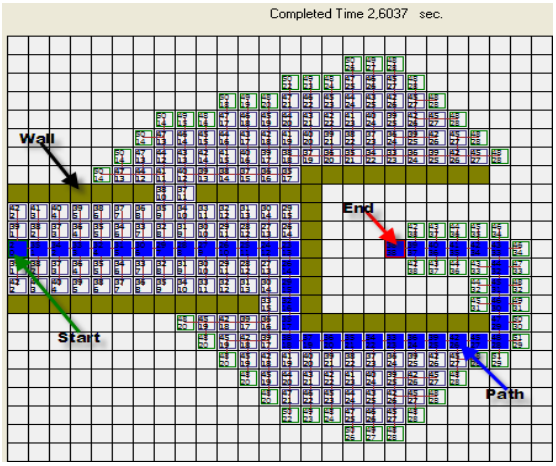

Figure 16: Path Planning around obstacles using A*

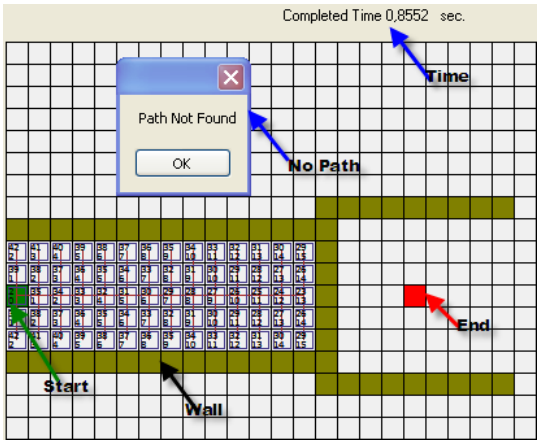

Figure 17: Path Planning when no path exist

In Figure 16, we observed that the manipulator will follow a straight path in blue towards the goal. While Figure 17 shows when no path exist between the start and the end point; otherwise, the manipulator would have meandered between the obstacles and walls (avoiding them) and eventually push the particle towards the destination.

5. Conclusion

In conclusion, we observed how the window application prepared using object oriented programming set of commands is used to achieve autonomous manipulation of objects in virtual space. Experiments regarding the implementation of conventional euclidean distance algorithm were demonstrated; in this case the solution for planning problem is trivial since the particles are considered to be having same physical properties. Therefore, the particle which is closest to the goal is the one which is pushed towards its destination. However, the application of artificial potential fields control algorithm is necessary in cases where the physical properties of the particles are considered to be different. The artificial potential fields' algorithm was implemented with modifications in which the manipulator only reacts by avoiding obstacles in its proximity. Further, application of motion planning algorithm A* was demonstrated with feature which facilitate the object avoidance in manipulaton of microobjects from one point to another in the virtual space. Since the objective is to reduce human intervention in the semi-autonomous manipulation, the addition of motion planning algorithms brings valuable ingredient to the MAW setup. All in all, this type of work platform can be seen as a step in using a standard platform in the manipulation of micro-entities using micro-scale manipulation tools.

Future work, include the implementation of similar motion planning algorithm in MAW setup.

Acknowledgement

We would like to express our appreciation to all other researchers at Sabancı University who were kind enough to share their views with us and offered some suggestions in making this project a success. Also, we would like to express our deepest appreciation to Yousef Jameel Scholarship Foundation of Berlin, Germany for the financial support in conducting this research.

References

[1] C. Belta, "Symbolic Approaches for Robot Motion Planning and Control." *ICRA'07,* Rome, Italy.

[2] J. H. Makaliwe and A. A. G. Requicha, "Automatic Planning of Nanoparticle Assembly Tasks", Proc. of the IEEE Int. Symposium on Assembly and Task Planning, pp. 288-293, Fukuoka, Japan, May 2001.

[3] C. Pawashe, and M. Sitti, "Two-Dimensional Vision Based Autonomous Microparticle Manipulation using Nanoprobe", Journal of Micromechatronics, September 2006.

[4] F. Markus Jönsson, "An optimal pathfinder for vehicles in real-world digital terrain maps", Master's Thesis, Royal Institute of Science Stockholm, Sweden, 1997.

[5] M. Kloetzer and C. Belta, "Managing non-determinism in symbolic robot motion planning and control." *ICRA'07,* Rome, Italy.

[6] A. Patel, "Amit's Game Programming Site," http://www-cs-students.stanford.edu/~amitp/gameprog.html, Accessed May, 2007

[7] N. Sabanovic, "Planning of One Part Movement", Project EACF05_00268, Istanbul, Turkey, December 2005.

[8] E.D. Kunt, "Design and Realization of a Microassembly Workstation," Masters Thesis, Sabancı University, Istanbul, Turkey, 2006

[9] S. Khan, A. O. Nergiz, A. Sabanovic and V. Patoglu, "Development of a Micromanipulation System with Force Sensing", IEEE IROS-2007.

[10] "A* algorithm implementation in C#" (updated May 2006), www.codeproject.com , Accessed June, 2007

[11] M. Bastan, "Visual Servoing of Mobile Robots Using Potential Fields", Masters Thesis, Sabanci University, Istanbul, 2004.

[12] V. I. Utkin, J. Guldner, J.Shi – Sliding Mode Control in Electromechanical Systems CRC Press, 1999.