METHODOLOGIES FOR IMMERSIVE ROBOT PROGRAMMING IN AN AUGMENTED REALITY ENVIRONMENT

S.K. Ong[2], J.W.S. Chong[1] and A.Y.C. Nee[1,2]
[1]Singapore-MIT Alliance
[2]Mechanical Engineering Department, National University of Singapore
9 Engineering Drive 1, Singapore 117576

Abstract

Advancements in robotics have gained much momentum in recent years. Industrial robotic systems are increasingly being used outside the factory floor, evident by the growing presence of service robots in personal environments. In light of these trends, there is currently a pressing need of identifying new ways of programming robots safely, quickly and more intuitively. These methods should focus on service robots and address long outstanding Human-Robot Interaction issues in industrial robotics simultaneously. In this paper, the potential of using an Augmented Reality (AR) environment to facilitate immersive robot programming in unknown environments is explored. The benefits of an AR environment over conventional robot programming approaches are discussed, followed by a description of the Robot Programming using AR (RPAR) system developed in this research. New methodologies for programming two classes of robotic tasks using RPAR are proposed. A number of case studies are presented and the results discussed.

CR Categories: H. 5.1 [Information interfaces and presentation (*e.g.* HCI]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I. 2.9 [Artificial intelligence]: Robotics—Operator interfaces; J. 2 [Computer applications]: Physical Sciences and Engineering—Engineering

Keywords: Robot programming, augmented reality, human-robot interaction, collision-free volume, path search, curve learning Bayesian neural networks, re-parameterization

1 Introduction

Current robot programming approaches lack the intuitiveness and efficiency required for quick and simple applications. There are three types of traditional robot programming methods, namely, lead-through, walk-through and offline programming [OSHA 2006]. These methods have a number of disadvantages. The first two methods, also known as online programming, pose safety concerns for the user who might be within the robot's workspace. The walk-through method is only suitable for certain types of robots because the actuator brakes need to be disengaged. The lead-through method using a teaching pendent is slow and unintuitive. Virtual Reality (VR) is an example of an offline method aimed at increasing the intuitiveness of the programming task for the user in a safe environment. Nanotek et al. [1995] and Aleotti et al. [2004] described VR systems used for the training of robots for object manipulation in environments that are known *a-priori*. However, totally-immersive environments like CAVE™ are costly and complex. Desktop-PC-based offline programming, on the other hand, is less intuitive as compared to the lead-through and walk-through methods. In general, offline programming is inflexible to unknown environments due to the need to model the physical entities in these environments, and calibrate and fine tune these simulated environments. Typically, offline programming packages also require the users to master complex programming languages that are software specific and thus, non-generic.

In an Augmented Reality (AR) environment, computer-generated objects are registered onto the real world view to enhance the user's real-time interaction with the real world. AR has been successfully applied in fields such as maintenance, assembly and computer-assisted surgery [Azuma 1997; Schwald and Laval 2003]. Relatively little work has been reported on the application of AR to robot programming. Most of the reported works are focused on telerobotics where the human does not directly interact with the robot [Rastogi and Milgram 1995; Pettersen et al. 2003].

This paper proposes the application of an AR environment for *immersive* robot programming where the user directly interacts with and controls a virtual robot amongst real entities in a real environment. Figure 1 shows a few examples of a virtual robot placed among real entities in miniature environments. The Robot Programming using AR (RPAR) approach is possible because the virtual robot is not subject to mechanical and physical constraints. Using a virtual robot means that programming tasks can be carried out safer as compared to traditional online programming methods. The approach proposed in this research is similar to the fast and intuitive walk-through method as it enables a human to directly move a virtual robot when planning a path. An AR environment can be created at the actual work cell, providing the flexibility and adaptability for different environments. This is because the entities in these environments need not be modeled and careful calibration between the various entities in the environments is not required. Without having to transport a real robot to the actual work cell, RPAR allows the evaluation of various robotic options before selection. Here, the user is able to evaluate simulations of planned paths for different virtual robots. If a robot has been selected, programming can be performed during the shipment period before the physical robot arrives at the actual work cell. Another potential application of an AR environment with a virtual robot is the programming of large robots using scaled-down models of the environment with the virtual robot, such as large pick-and-place robots.

New RPAR methodologies have been developed for two classes of tasks in this research. Class I tasks have a number of possible path solutions for a given start and goal points, such as general pick-and-place, spot welding and inspection tasks. Class II tasks constrain the end-effector to follow a user-defined path, at a certain orientation with respect to that path, such as arc welding

GRAPHITE 2006, Kuala Lumpur, Malaysia, November 29 – December 02, 2006.
© 2006 ACM 1-59593-564-9/06/0011 $5.00

and laser cutting. These two classes of tasks are selected because the focus of the current work is on relatively simple non-contact tasks. Therefore, for Class I tasks, physical manipulations of the objects are not considered or are assumed to have been performed prior to the planning of the collision-free path. The developed methodologies complement the notion of Programming by Demonstration [Kaiser et al. 1995; Friedrich et al. 1996] where a human operator is required in the path planning activity to perform certain tasks that are difficult to be simulated in a computer, such as recognizing and interpreting an environment and providing appropriate data via demonstrations.

The remaining of this paper is organized as follows. Section 2 describes the RPAR system. Section 3 presents the proposed methodologies for programming robots for Class I tasks and Section 4 presents the methodologies for Class II tasks. Section 5 presents the results and discussions of RPAR applied to a number of case studies for both classes of tasks. Lastly, Section 6 concludes and proposes future research opportunities.

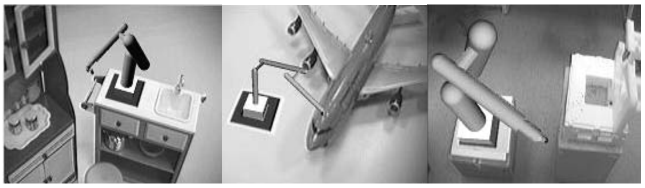


Figure 1: Virtual robots in real environments: (a) miniature general-purpose robot in a miniature kitchen, (b) miniature airplane washing robot, and (c) a full-scale robot model.

2 RPAR System Description

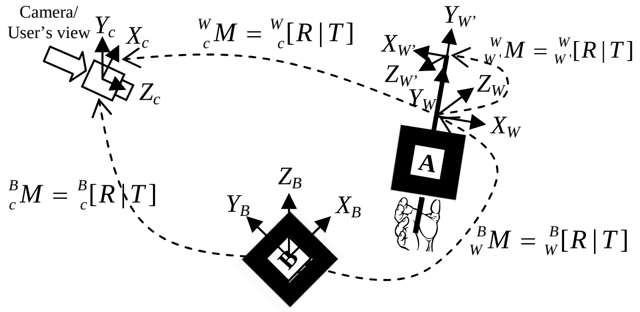
2.1. Relationships between Coordinate Systems

There are three major Euclidean coordinate systems in the RPAR system, namely, the camera, the robot base and the robot wrist, which is a handheld probe used by the user to control and move the virtual robot. For example, in Figure 2, the **B** marker is used as the base and the **A** marker as the wrist. The relationships between the camera (X_c, Y_c, Z_c), robot base (X_b, Y_b, Z_b) and the wrist (X_W, Y_W, Z_W) coordinate systems are obtained using the marker detection method in ARToolkit [Kato and Billinghurst 1997], while the relationship between the wrist and the end-effector reference point ($X_{W'}$, $Y_{W'}$, $Z_{W'}$) coordinate systems is known (the end-effector reference point can also be the tip of the handheld probe). Figure 3 shows examples of different handheld probes with miniature end-effectors. The matrix M relates any two coordinate systems and consists of the 3×3 rotation, R and 3×1 translation, T matrices. Thus, the relationship between the robot base, B and the end-effector reference point, W' frames is,

$$ {}^{B}_{W'}M = {}^{B}_{c}M[{}^{W}_{c}M]^{-1}\,{}^{W}_{W'}M = {}^{B}_{c}M\,{}^{c}_{W}M\,{}^{W}_{W'}M \qquad (1)$$

2.2. Robot Kinematics Model

The configuration of a robot is characterized by its generalized coordinates, $\mathbf{q} = [q_1, q_2, .., q_n]$, where n is the number of degrees-of-freedom (dof). To illustrate the methodology, a six dof revolute joint robot as shown in Figure 4 is used (similar to a PUMA 560 industrial robot with three axes intersecting at the wrist [Craig 1989]). The angles of the joints are denoted as $\mathbf{q} = \boldsymbol{\theta} = [\theta_1, \theta_2, .., \theta_6]$. Figure 4 shows an example of a rendered gripper-type end-effector corresponding to a physical probe with the same end-effector type. It should be noted that other robot models can be used, thus providing the flexibility to evaluate various robotic options. The matrix ${}^{B}_{W'}M$ that relates the end-effector reference point to the base is the kinematics chain of the robot model that has to be solved. The kinematics analysis can be divided into forward and inverse kinematics. The forward kinematics analysis is straightforward, where given the configuration of the robot, the unique position and orientation of the end-effector can be determined using the Denavit-Hartenberg convention [Craig 1989]. The inverse kinematics of a robot is much more complex as there are multiple solutions by which the robot can achieve a given end-effector orientation and position. However, setting allowable joint ranges or using the geometric approach (only one arm configuration type is picked at a time) avoids multiple solutions and unwanted configurations, such as singularities where the robot loses a dof. Avoiding multiple solutions at any one time is essential to ensure that the process is intuitive for a human who is moving the robot. If the end-effector is moved out of the workspace of the robot, the robot will stop and remain at the last feasible configuration prior to the workspace violation. Both forward and inverse kinematics solutions that have been reported by Craig [1989] are used in the RPAR system.


Camera/ User's view
Y_c
X_c
Z_c
${}^{W}_{c}M = {}^{W}_{c}[R|T]$
$Y_{W'}$
$X_{W'}$
$Z_{W'}$
${}^{W}_{W'}M = {}^{W}_{W'}[R|T]$
Y_W
Z_W
X_W
A
${}^{B}_{c}M = {}^{B}_{c}[R|T]$
Z_B
Y_B
X_B
B
${}^{B}_{W}M = {}^{B}_{W}[R|T]$
Figure 2: Relationships between main coordinate systems.

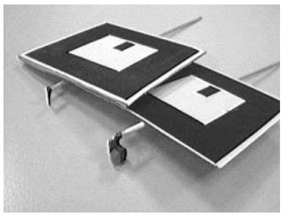


Figure 3: Handheld probes with different end-effectors.

2.3. RPAR System Architecture

The basic RPAR system architecture is shown in Figure 5. The system was implemented using the video-based tracking method in ARToolkit and the C programming language. The input to the inverse kinematics module is the pose of the marker of the wrist, which can be obtained using the pose determination module in ARToolkit. The position and orientation of the handheld probe can also be calculated. The inverse kinematics module calculates the necessary generalized coordinates of the virtual robot based on the obtained pose, while the forward kinematics module calculates the coordinates of the end-effector reference point based on the generalized coordinates. For each video frame, a display of the virtual robot and other virtual elements is provided to the user through the HMD (Figure 6), using the OpenGL Renderer. The

HMD of the RPAR system consists of a single IEEE Firefly camera and an i-glasses video display goggle. This virtual display serves as feedback that is useful for users' interaction and actions, and the evaluation of the outcomes of these actions. The torque/force module calculates the amount of work or effort made by the robot for a particular movement. The path planner contains tools for planning collision-free paths, which will be transferred to the robot controller for further processing and execution. In this work, the physical robot controller is not considered.

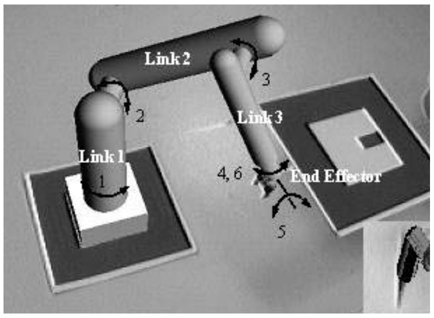


Figure 4: Robot model with an alternative end-effector.

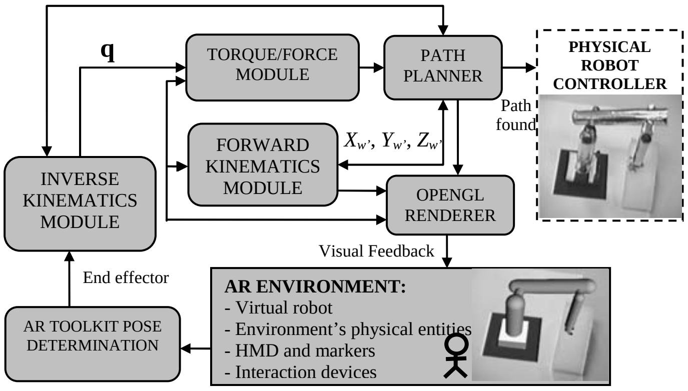


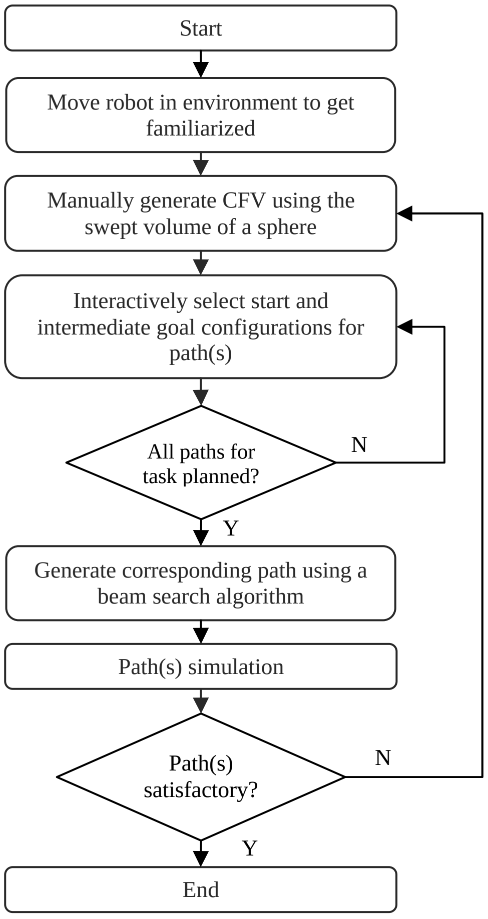
Figure 5: Current RPAR system architecture.

Figure 6: HMD used for visualization.

3 Proposed Methodology for Class I Tasks

3.1. Proposed Methodology

The proposed methodology is shown in Figure 7. In the first step, a user has to familiarize himself/herself with the moving of the robot in the environment. In the next step, the user manually generates a Collision-Free Volume (CFV) using the swept volume of a sphere attached to a probe (Section 3.2). The path(s) are then planned by interactively selecting valid start and intermediate goal configurations in a sequential manner. A beam search algorithm (Section 3.3) is used to automatically generate the corresponding collision-free path(s) for the robot joints (as opposed to a single direct demonstration). Next, this information is used to simulate the virtual robot performing the path(s) in the environment, for the user to visually evaluate the smoothness of the path(s) and for any potential collisions. If the results are unsatisfactory, the user may choose to repeat the process.


Start
Move robot in environment to get familiarized
Manually generate CFV using the swept volume of a sphere
Interactively select start and intermediate goal configurations for path(s)
All paths for task planned?
N
Y
Generate corresponding path using a beam search algorithm
Path(s) simulation
Path(s) satisfactory?
N
Y
End
Figure 7: Proposed methodology for Class I tasks.

The proposed approach incorporates the ability of the human to plan collision-free paths quickly. Through moving the end-effector probe, the user is able to move the virtual robot in an environment that consists of real entities to perform path planning. A CFV, which defines a subset of the entire free space that is relevant to the task, is manually generated in a manner similar to an artist 'painting on a 3D canvas'. This approach is different from the use of multiple direct demonstrations to define the free space [Delson and West 1994a; 1994b]. The CFV allows the search for paths to focus only at areas deemed important for the tasks. Unlike offline planners, the CFV avoids the need to model the environment entities as this information is implicit in the CFV. Therefore, the robot would not collide with any obstacles in the environment if it remains within the volume.

The methodology aims to generate smooth paths consistently and more reliably as compared to paths demonstrated directly by a human. Therefore, the use of a *single* direct demonstration (using the end-effector probe to move the robot along a path) as the final path, is undesirable. This is because the resolution and smoothness of the generalized coordinate profiles for the resulting path will be inconsistent due to either one or a combination of (1) inconsistency in the hand movements, (2) inconsistency in the sampling rate, and (3) a low sampling rate. Although smoothing can be performed as a post-processing step, it is difficult to determine the appropriate level of smoothness so as to ensure that the path is collision-free [Delson and West 1994a]. For these reasons, collision-free paths that are constrained by the CFVs, which are generated by the users, are automatically generated using a beam search algorithm.

3.2. Collision-Free Volume

The CFV is generated by dragging a sphere that is attached to the end of the handheld probe (Figure 7). It consists of a set of *p* virtual spheres,

$$\text{CFV} = \{ S_i(c_i, r_0);\ i = 1,..,p \} \in \Re^3 \qquad (2)$$

where S_i is a virtual sphere, characterized by a centre point, c_i, and a radius, r_0.

The 3D coordinates of these centre points are determined with respect to the world coordinate system, which is the coordinate system of the base of the robot (**B** marker in Section 2.1). The choice of the radius of the physical sphere, *r* depends on a number of factors. In practice, the physical sphere used should be larger than the virtual sphere, which is the basic unit of the CFV. Therefore, *r* is defined by,

$$r = r_0 + (\Delta r_{robot\_error} + \Delta r_{tracking} + \Delta r_{modeling}) \qquad (3)$$

where Δr_{robot_error} is the maximum error of the physical robot, $\Delta r_{tracking}$ is the maximum error of the tracking process, and $\Delta r_{modelling}$ is the modeling error.

The selection of *r* depends on the level of resolution required of the environment (whether the sphere is able to access smaller spaces). A combination of different sphere radii can be used. Using an AR environment, various options for visual feedback can be employed. The display of the CFV is semi-transparent to allow the user to view the overall environment at all times, and enable a visual inspection of the CFV generated. The virtual robot is rendered during the CFV generation, as shown in Figure 8(b), to enable the user to observe whether the robot is able to reach all sections of the CFV, and check for collisions between the less critical parts of the robot (for example, links 1 and 2) with any obstacles in the environment.

The quality of the CFV can be evaluated based on its compactness and regularity in shape. The issue of interpolating between samples does not need to be considered in the CFV generation although the sampling rate for this video-based system is low. This is because, although interpolation may be useful to obtain a more condensed CFV, the resulting CFV may not be collision-free, as illustrated in Figure 9. Other ways to obtain a good CFV include using a faster sampling rate or moving the sphere at a slower speed. An adequate CFV can be obtained at a low frame rate of 10 frames per second (fps) if the user avoids fast and sudden movements. Lastly, the user can perform multiple passes to generate a more condensed and regular volume.

After the CFV has been defined, a *CFV check* is performed to determine whether critical parts of the robot are within this volume and hence collision-free. The choice of critical parts depends on the application at hand. In this research, the focus is on the end-effector as it is essentially the part of a robot that is most likely to collide with obstacles. Therefore, it requires a more thorough check as compared to other parts of the robot. The *CFV check* is performed through verifying whether a bounding volume of the end-effector is within the CFV. In this work, a bounding cylinder is used because it is a tighter bound as compared to a bounding box and it is also more volume efficient.

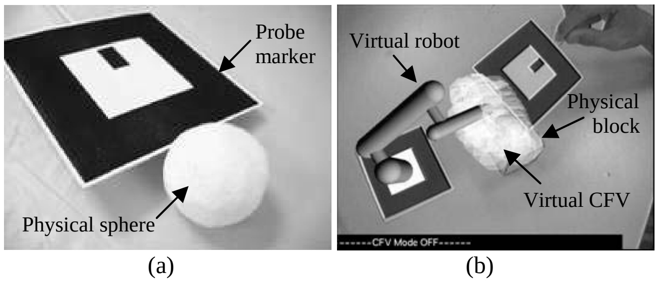


Figure 8: (a) Sphere attached to a probe with marker, and (b) generating a CFV over a rectangular block.

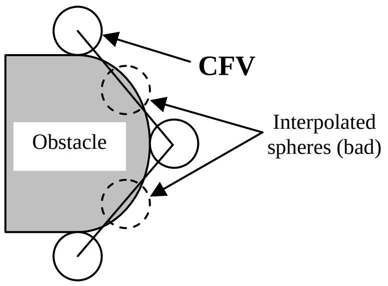


Figure 9: Interpolating between spheres.

3.3. Collision-Free Path Generation

After the CFV has been generated, the user can directly define or demonstrate the start and goal configurations. Tasks are planned by selecting valid start-goal configuration pairs in a sequential manner. Valid configurations are those that are within the workspace of the robot, and within the CFV. If the end-effector moves outside the CFV, the particular configuration involved is invalid or infeasible and a warning will be shown.

In this work, a beam search strategy [Furcy 2004] is adopted to generate the paths. The algorithm maintains at most *k* paths in its search for the goal. Beginning from the start configuration, the algorithm selects at most *k* best nodes from a neighborhood region that is created by perturbing the latest nodes of the paths maintained. Before the selection, the neighborhood region is filtered by removing nodes that are invalid. The best nodes are selected using a heuristic called *dist-to-goal*, which indicates how close a particular configuration is to the goal configuration (best nodes are nodes with the lowest *dist-to-goal* value). For the six dof robot in Section 2.2,

$$\textit{dist-to-goal} = \sqrt{\sum_{j=1}^{n} (q_{j,t} - q_{j,G})^2} \qquad (4)$$

where $q_{j,t}$ is the configuration for joint *j* at search step *t* and *G* is the goal node.

In practice, as the size of the beam *k* increases, the quality of the solution and robustness of the search improve. The goal is found when for any latest node's configuration,

$$\left| q_{j,t} - q_{j,G} \right| \le \varepsilon_j, \qquad \in j = 1,..,n \qquad (5)$$

where ε_j is the fixed perturbation magnitude for each joint.

When two or more paths arrive at the same configuration at a particular search step, only the path with the lowest cost is kept and maintained. For simplicity, the path cost in this work is determined by calculating the gravitational torque at each joint without considering the robot's dynamics. This is the decoupled trajectory learning approach where the geometric path is first planned followed by the velocities and accelerations for motion [Bobrow et al. 1989]. However, the dynamics model of the robot may also be included as the search method is independent of the path cost formulation.

4 Proposed Methodology for Class II Tasks

4.1. Proposed Methodology

Figure 10 shows the proposed methodology to generate a collision-free path for a robot, where the end-effector is required to follow a visible desired curve, which position is unknown, at a consistent orientation with respect to the curve. In the first step, the user is required to perform a number of repeated and separate demonstrations of the desired curve to be followed so as to collect data points. Visual feedback for the user's evaluation is provided as the data points are generated. In the second step, known as *learning* the curve, the data is processed using a Bayesian neural network to generate an output curve that is close to the desired curve. After the output curve has been obtained, the user may familiarize himself/herself with the moving of the robot in the environment. In the next step, a CFV is manually generated around the areas relevant to the curve. This CFV is needed to verify that the robot's end-effector does not collide with any obstacles along the curve. Next, the orientation of the end-effector with respect to the curve is planned interactively by the user using the AR interface. To verify that a path (in terms of the robot's generalized coordinates) is indeed collision-free, the path is simulated using the calculated end-effector transformations along the curve. If the end-effector cannot be guaranteed to be within the CFV (hence, collision-free) at all times, adjustments are made to the end-effector's previously selected orientation until a successful geometric path is found.

4.2. Curve Learning and Generation

The desired path/curve that has to be followed by the robot is represented using the parametric form [Sung 2004],

$$\begin{pmatrix} X(\mathbf{a},t) \\ Y(\mathbf{a},t) \\ Z(\mathbf{a},t) \end{pmatrix} \text{ for a 3D space curve.} \qquad (6)$$

where X, Y, and Z are 3D coordinates, $\mathbf{a}$ is the vector of model parameters and t is the location parameter (*e.g.* time).

The data points for the unknown curve are obtained through dragging the tip of the handheld probe (Figure 11(a)) over the visible desired curve (Figure 11(b)). Visual feedback using AR enables the user to reject demonstrations that are bad, *i.e.*, demonstrations that are an obvious mismatch of the desired curve or contain many outliers. The data points from a number of good demonstrations (3 or 4 demonstrations) are then kept and parameterized using an algorithm that determines the closest location of a data point to a piecewise linear approximation of the desired curve. This approximation is manually obtained by the human through selecting a number of points while guided by visual AR overlays. An example is shown in Figure 11(c).

A Bayesian neural network and re-parameterization approach, as shown in Figure 12, are used to process the data set obtained from a number of good demonstrations. The network used consists of a single hidden unit layer with parameter t as the input and 3D coordinates $\left[X(\mathbf{a},t), Y(\mathbf{a},t), Z(\mathbf{a},t)\right]^T$ as the outputs. The goal is to produce an output curve that best represents the demonstration data, *i.e.*, learn the desired curve. The Bayesian framework can be incorporated into a neural network [MacKay 1995] through modifying the standard backpropagation error equation,

$$E(\mathbf{W}) = \beta E_D + \alpha E_{\mathbf{W}} \qquad (7)$$

where $\mathbf{W}$ is the vector of weights w_i, E_D is the sum of squared errors SSE for the network, $E_{\mathbf{W}} = \frac{1}{2}\sum_{\in i} w_i^2$ is the sum of squared weights SSW, and β and α are regularization parameters.

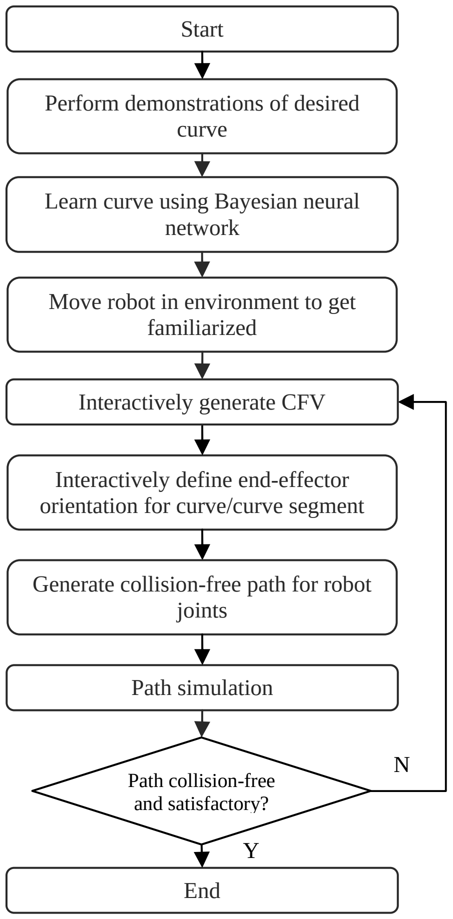

Start
Perform demonstrations of desired curve
Learn curve using Bayesian neural network
Move robot in environment to get familiarized
Interactively generate CFV
Interactively define end-effector orientation for curve/curve segment
Generate collision-free path for robot joints
Path simulation
Path collision-free and satisfactory?
N
Y
End

Figure 10: Proposed methodology for Class II tasks.

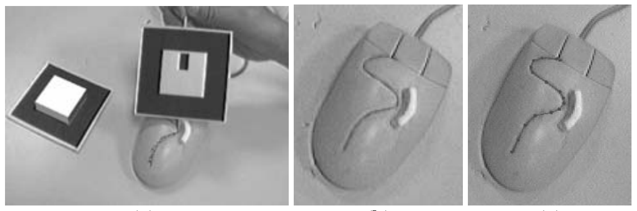


Figure 11: (a) Tracing a desired curve, (b) original desired curve, and (c) manually generated approximation of desired curve.

```
1.  Run Bayesian neural network until convergence
    using initial parameterization.
2.  Simulate network to obtain initial output
    curve.
3.  repeat
4.      Perform re-parameterization based on
        Equation (8).
5.      Run network for one iteration with new
        input-output set.
6.      Simulate network to obtain output
        curve.
7.  until convergence.
```

Figure 12: Bayesian neural network and re-parameterization algorithm

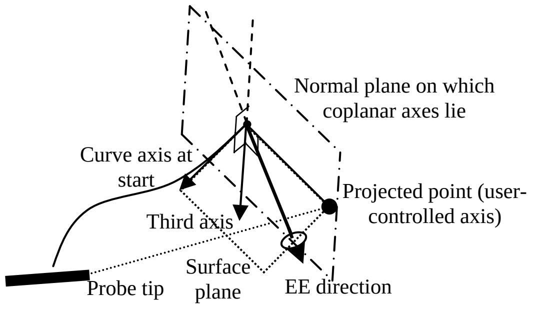

Normal plane on which coplanar axes lie
Curve axis at start
Projected point (user-controlled axis)
Third axis
Surface plane
Probe tip
EE direction

Figure 13: Example of defining the first coordinate system and the orientation of the end-effector at the start of the curve.

Re-parameterization is a concept used in standard fitting problems where the parameters for the data points are updated *after* each least-squares iteration [Hoschek 1988; Sung 2004]. This is to ensure that the Bayesian neural network learns the curve based on orthogonal distances. The initial output curve of the Bayesian neural network is used as a good initial guess for the re-parameterization steps. Re-parameterization is performed using *point-to-point* matching [Besl 1991]. If a particular curve is given as a set of parameterized points, the orthogonal distances between the data points and the curve points can be found. The shortest Euclidean distance between a particular measurement point $\mathbf{X}_i$ and a curve F that consists of p points, $\mathbf{X}_j^F$ is defined as,

$$l_i = \min_{j \in (1,..,p)} \left\| \mathbf{X}_i - \mathbf{X}_j^F \right\| \tag{8}$$

After each iteration, the parameters of the data points are updated based on Equation (8), which determines the closest location on the current output curve for each data point. Curve learning is performed by adjusting the complexity of the neural network using the Bayesian method after each step. Convergence is achieved when the SSE, SSW or number of effective parameters used does not change significantly over a few iterations.

The parametric form was selected due to its flexibility as opposed to the implicit and explicit forms which are dependent on the choice of the coordinate system [Farin 2004; Sung 2004]. A neural network approach was chosen because the model of the desired curve (**a** in Equation (6)) in the problem addressed in this research is unknown and needs to be learnt. This is the major difference between the standard fitting problem and the learning problem addressed here. The parameters **a** for the model that has been learnt are the final values of the effective weights in the neural network.

4.3 Collision-Free Path Generation

After the output curve is obtained, the user first generates a CFV around the areas relevant to the output curve. This will be used to verify if the end-effector is collision-free along the curve. Next, the end-effector orientation with respect to the curve can be planned interactively with the aid of AR. The user is required to define the coordinate system at the beginning of the curve, where one axis, *curve-axis*, is set to be in the direction of the curve. Another axis is defined by the user by projecting the probe tip onto the plane normal to the *curve-axis* as shown in Figure 13. A third axis that is co-planar with the user-controlled axis is thus uniquely defined. A pre-determined transformation from the coordinate system of the curve to the robot's wrist is used to obtain the initial transformation from the robot's base to the end-effector. AR is used to display the direction of the end-effector's principal axis (cone in Figure 13) corresponding to the defined coordinate system at the start of the curve. Based on the user's judgments on the surface on which the curve lies (surface plane) and the process requirements, a desired orientation is selected.

The other curve coordinate systems are next obtained by determining the transformations that adjust and align the *curve-axis* to the output curve, moving from the start to the end of the curve. Each curve transformation is multiplied with the pre-determined transformation to the wrist to obtain the end-effector's relationship to the robot's base. These will be the inputs to the inverse kinematics module that will generate the simulation path.

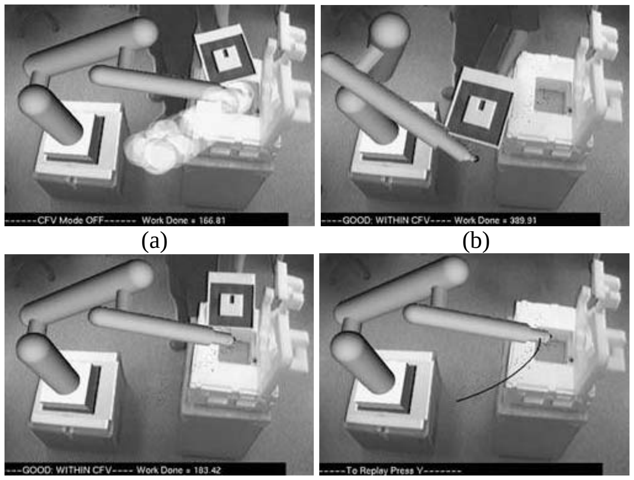
5 Results and Discussions for Case Studies

5.1. Case Studies for Class I Tasks

A number of pick-and-place tasks were planned in a full-scale environment (Figure 14) and in a miniature environment (Figure 15). After the user has generated the CFV at the areas relevant to the tasks, the beam search algorithm is used to automatically generate the collision-free paths. For the full-scale environment, a single collision-free path was planned. For the miniature environment, two sequential paths were planned. The end-effector is moved from a start configuration to an intermediate configuration on a table in the first path, and from the intermediate configuration to the final goal configuration on a conveyor belt in the second path. As the user moved the virtual robot, the inverse kinematics and the *CFV Check* modules indicate if a particular configuration is valid. For the full-scale case, screen-based visualization was used as opposed to the HMD.

5.2. Case Studies for Class II Tasks

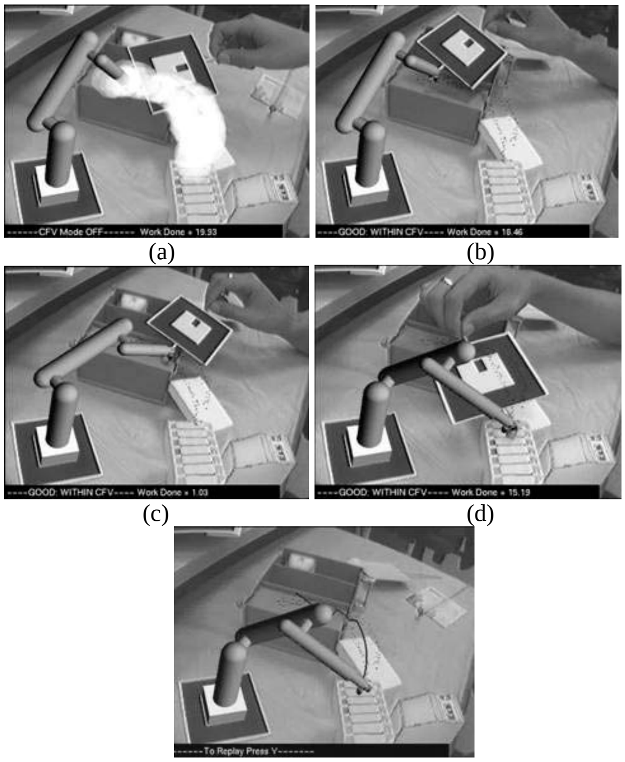
The qualities of the output curves were evaluated by comparing the curves to *known* desired curves. It was observed that in general, better quality output curves can be obtained with an increased number of demonstrations. This is due to the availability of more data points, which increases the probability of having good data points. The improvement becomes less significant as more demonstrations are used. In addition, with more demonstrations, the distribution of the errors becomes increasingly normal with respect to the desired curve. This is important because the proposed method using Bayesian neural networks in Figure 12 [MacKay 1995] is based on the assumption that the errors are normally distributed. Figure 16 shows a case study of a miniature 'AR' curve on a planar surface, and Figure 17 shows a full-scale curve on a piece of sheet metal. The output curves were learnt based on three good demonstrations.


------CFV Mode OFF------ Work Done = 166.81
----GOOD: WITHIN CFV---- Work Done = 389.91
---GOOD: WITHIN CFV---- Work Done = 183.42
-----To Replay Press Y-------

(a) (b)

(c) (d)

Figure 14: Class I task in a full-scale environment: (a) generating a CFV, (b) selecting start configuration, (c) selecting goal configuration, and (d) final path found.


------CFV Mode OFF------ Work Done = 19.93
----GOOD: WITHIN CFV---- Work Done = 18.46
----GOOD: WITHIN CFV---- Work Done = 1.03
----GOOD: WITHIN CFV---- Work Done = 15.19
------To Replay Press Y-------

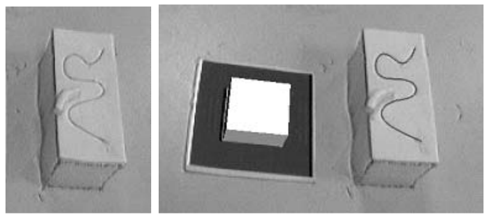
(a) (b)

(c) (d)

(e)

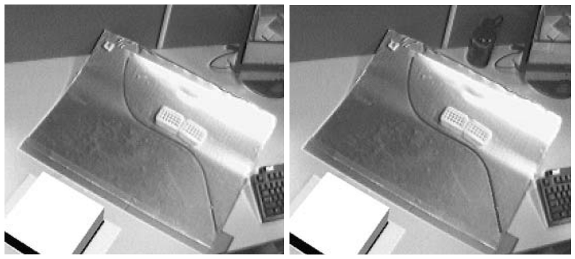
Figure 15: Class I task in a miniature environment: (a) generating a CFV, (b) selecting start configuration, (c) selecting intermediate goal configuration, and (d) selecting final goal configuration, and (e) final paths found.

The planning of the end-effector orientation along the output/learnt curve using AR is shown in the full-scale case study (Figures 18 and 19). The user first moved the virtual robot around the environment for familiarization and performed a visual evaluation on its reachability (Figure 18(a)). Next, a CFV is generated along the curve (Figure 18(b)). Next, the user selected a desired end-effector orientation using the user-controlled *X*-axis, where the arrow (not part of original picture) shows the projection of the probe tip onto the normal plane, and the cone shows the end-effector direction (Figure 18(c)). In this work, a right-hand notation is used where the user-controlled axis, *curve-axis* and the third axis are set as the *X*-axis, *Y*-axis and *Z*-axis respectively. The top left corner of Figure 18(c) shows a magnified view of the region concerned. Finally, a successful collision-free path, where the end-effector does not collide with the white obstacles along the curve, is simulated, as shown in Figure 18(d) and Figure 18(e). The angular profiles are shown in Figure 19.



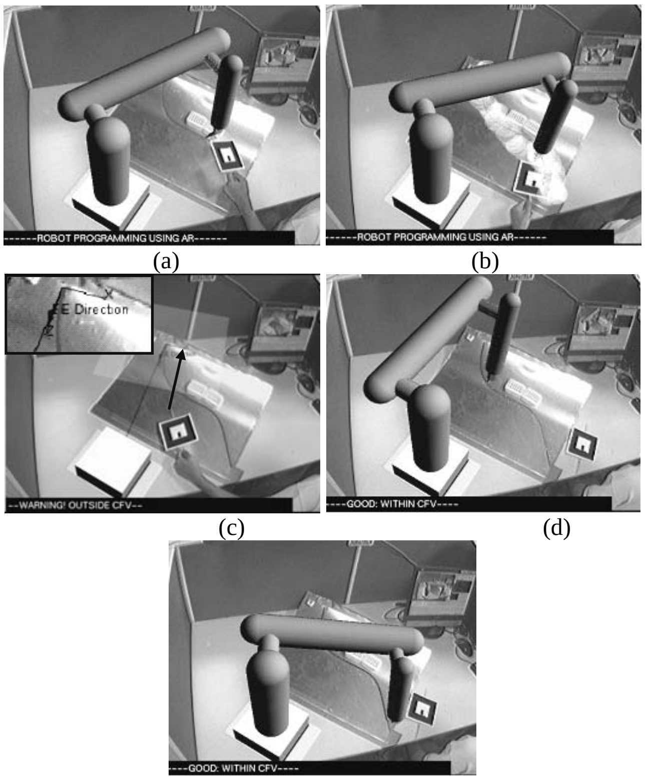
(a) (b)

Figure 16: Output 'AR' curve (b) superimposed on original desired curve (a) on a planar surface.



(a) (b)

Figure 17: Output curve (b) superimposed on the original full-scale desired curve (a) on a piece of sheet metal.



(a) (b)

(c) (d)

(e)

Figure 18: Full-scale planning of the end-effector orientation.

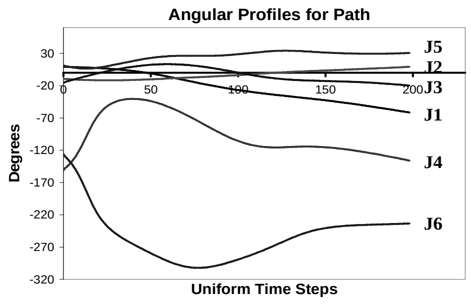


Figure 19: Angular profiles of joints for path planned in Figure 18.

5.3. Discussion

For both classes of tasks, the RPAR system utilizes the ability of the human to provide appropriate domain knowledge of an environment using an AR interface. The level of intuition is high as the user is made to feel that he/she is actually 'guiding' a real robot similar to the walk-through programming approach. For example, selecting a desired configuration is as simple as moving the virtual robot to a configuration and selecting it, *i.e.*, demonstrating the user's intent to the RPAR system. Tasks that are difficult for humans, such as generating smooth paths and learning curve models, are handled by the RPAR system. Hence, the methodologies establish the individual roles of the humans and computers for the robot programming and planning process.

6 Conclusions

In this paper, a RPAR system for immersive robot programming, where the user directly moves a virtual robot in an unknown environment is presented. The strengths of AR over conventional methods include flexibility in providing visual guidance to the user during the programming process, flexibility to various environments without the need to model the environment entities, and increased intuition and efficiency in the robot programming process. Two robot programming methodologies for two classes of tasks were proposed. The first methodology targets applications where there are a number of possible path solutions for a given start-goal configuration pair. The methodology is generic, and different path search algorithms other than the beam search strategy proposed in this paper can be employed. The second methodology targets applications where the end-effector is required to follow a pre-determined 3D path/curve at a consistent orientation with respect to the curve. The end-effector orientations are planned interactively with the aid of AR visualization. Both methodologies involve the generation of a CFV, which is needed to verify that the final geometric path is collision-free. Future research opportunities include the development of improved methods for generating and representing the CFV, more efficient algorithms for verifying objects are within the CFV, and the design of suitable path search and generation algorithms. Another issue that needs to be investigated is the level of accuracy achievable using RPAR. It is expected that the use of more sophisticated and accurate tracking systems would significantly improve user acceptance of these methodologies.

References

ALEOTI, J., CASELLI, S., AND REGGIANI, M. 2004. Leveraging on a Virtual Environment for Robot Programming by Demonstration. *Robotics and Autonomous Systems 47*, 2-3, 153-161.

AZUMA, R.T. 1997. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments 6*, 4, 355 – 385.

BESL, P.J. 1992. A Method for Registration of 3D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2, 239-256.

BOBROW, J.E., DUBOWSKY, S., AND GIBSON, J.S. 1985. Time-Optimal Control of Robotic Manipulators along Specified Paths. *International Journal of Robotics Research 4*, 3, 3-17.

CRAIG, J.J. 1989. *Introduction to Robotics, Mechanics and Control*. Addison-Wesley.

DELSON, N., AND WEST, H. 1994a. Robot Programming by Human Demonstration: The Use of Human Variation in Identifying Obstacle Free Trajectories. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 564-571.

DELSON, N., AND WEST, H. 1994b. Robot Programming by Human Demonstration: The Use of Human Variation in Improving 3D Robot Trajectories. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, 1248-1255.

FARIN, G. 2002. *Curves and Surfaces for CAGD: A Practical Guide, 5th Ed.* Morgan-Kaufman.

FRIEDRICH, H., MÜNCH, S., DILLMANN, R., BOCIONEK, S., AND SASSIN, M. 1996. Robot Programming by Demonstration (RPD): Supporting the Induction by Human Interaction. *Machine Learning 23*, 2-3, 163-189.

FURCY, D. 2004. *Speeding Up the Convergence of Online Heuristic Search and Scaling Up Offline Heuristic Search*. PhD thesis, Georgia Institute of Technology.

HOSCHEK, J. 1988. Intrinsic Parametrization for Approximation. *Computer Aided Geometric Design 5*, 1, 27-31.

KAISER, M., RETEY, A., AND DILLMANN, R. 1995. Robot Skill Acquisition via Human Demonstration. In *Proceedings of the 7th International Conference on Advanced Robotics (ICAR)*, 763-768.

KATO, H., AND BILLINGHURST, M. 1999. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality,* San Francisco, 85-94.

MACKAY, D.J.C. 1995. Bayesian Methods for Neural Networks: Theory and Applications. Lecture Notes, University of Cambridge Programme for Industry, July.

NANOTEK, E., ZIMMERMAN, T., AND FLUCKIGER, L. 1995. Model Based Vision as Feedback for Virtual Reality Robotics Environment. In *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, 110-117.

PETTERSEN, T., PRETLOVE, J., SKOURUP, C., ENGEDAL, T., AND LOKSTAD, T. 2003. Augmented Reality for Programming Industrial Robots. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 319-320.

RASTOGI, A., AND MILGRAM, P. 1995. Augmented Telerobotic Control: A Visual Interface for Unstructured Environments. In *Proceedings of the KBS/Robotics Conference*, 16-18.

SCHWALD, B., AND LAVAL, B. 2003. An Augmented Reality System for Training and Assistance to Maintenance in the Industrial Context. *Journal of WSCG 11*, 1, 425-432.

SUNG, J.A. 2004. *Least Squares Orthogonal Distance Fitting of Curves and Surfaces in Space*. Springer-Verlag.

U.S. DEPARTMENT OF LABOR, 2006. OSHA Technical Manual, Industrial Robots and Robot System Safety, Section IV: Chapter 4. Available: http://www.osha.gov/dts/osta/otm/otm_iv/otm_iv_4.html, last accessed on 10 Sept. 2006.