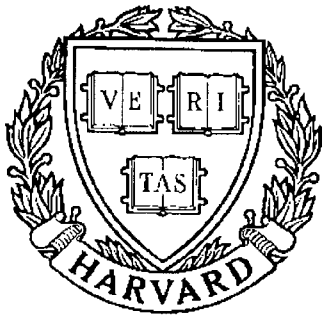
UNDERGRADUATE REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Optimal Path-Finding Algorithm for Mobil Robotics Devices

by C. Baker, T. Lu, M. O'Malley and B. Tegeler
Advisor: G.M. Zhang

U.G. 90-2

Optimal Path-Finding Algorithm for Mobile Robotics Devices

Undergraduate Students:

Craig Baker
Thomas Lu
Mike O'Malley
Bret Tegeler

Faculty Advisor:

Guangming Zhang

December 1990

Table of Contents

Abstract

1.1 Objective

Robotic technology is being developed to increase productivity and alleviate wasted time. The trajectories of motion which these robots operate on are determined by matrix algorithms within the controller of the robots. This report contains a detailed analysis of the optimal-path finding algorithm which can be integrated with a robotic manipulator.

1.2 Procedures

Using a two-dimensional model, an algorithm was written to input the coordinates of an infinite number of objects, the destination, and the location of the robot. The program calculates and optimizes the path required to pick up the objects and position them at the desired location in an optimal fashion. Utilizing the Graphics Kernal System (GKS), the path was then animated to visually demonstrate that the selected order to pick the objects was in fact the optimal one. The final aspect of the project involved the integration of the path-finding algorithm with a PUMA 500 series robot .

1.3 Conclusion

A successful optimal path-finding program was developed and implemented with the GKS software. By translating the program into the PUMA robot language, VAL-II, the algorithm was utilized in a practical and easily adaptable manner.

Introduction

2.1 A Chronological Perspective[1]

Robotic technology is fairly recent and it is only in the last 30 years that industry has adopted the robot as a viable alternative to numerically controlled machines and teleoperators. With the improvement of the computer and it's processing speed, the robot is slowly moving from the research sector to the industrial base. In the 1960's large firms such as automotive manufacturers began using robots to perform tedious and repetitive tasks. Thus, welding and spray painting still remain as the leading use of industrial robots today.

Ideal for highly repetitive tasks, robots conform to the manufacturing industry's needs. This particular industry's desire for time saving methods has spotlighted the demand for optimal-path finding algorithms. Time saving devices include improved mechanical elements such as bearings and fluid clutches to reduce friction and speed response time. Mathematical modeling of the kinematic and dynamic response of robotic manipulators are difficult, but are improving with increased efficiency of semi-conductors and microprocessors. Thus, with the mechanical elements modeled, an optimal path-finding algorithm can be accurately tested and integrated into a robotic work envelope.

2.2 The Algorithm

The algorithm to optimize the path taken by a robot to select an infinite number of objects and position them at a destination can be divided into three categories:

[1] Wolovich, William A., ROBOTICS: Basic Analysis and Design, CSC College Publishing, ©1987, pgs.4-17.

1. Input Module
2. Evaluation of total transportation distance for N objects
3. Determination of the shortest path

Written in the C programming language, the algorithm and sorting routines are not extremely complex and therefore can easily translated into other programming languages such as VAL-II, FORTRAN, and BASIC.

The input module prompts the user to enter the number of objects, the coordinates of the object, the coordinates of the destination, and the coordinates of each of the objects. A routine then calculates the distance traversed from the robot to the object and then to the destination. Also determined is the distance from the destination to the object and then back to the destination. The program then calculates the possible paths and the total distances that the robot has to travel. From here a sorting routine operates on the paths and distances to generate an optimal path. The output states the order in which the objects should be retrieved.

2.3 Graphics Kernal System

The Graphics Kernal System is composed of software to graphically illustrate mathematical manipulations. The system's usefulness lies in it's ability to simulate a robots trajectory or work space. This reduces the amount of time required for modeling and aids in the visualizing a problem.

The optimal path routine was simulated using GKS. The screen of the SUN computer was covered with a grid on which the robot, objects, and destination appeared. The robot appeared as a triangle, the objects as circles, and the destination as a square. When the robot moves to retrieve an object, its path was traced with a solid line. The object to be retrieved, a circle, then disappeared to demonstrate that the object was in fact retrieved. Thus, the GKS program aided in the visualization of the algorithm. Written in the C programming language, the GKS portion of the project interlock with the optimizing algorithm. The ease of interfacing made GKS a helpful tool in analyzing the kinematics of the program.

2.4 PUMA 500 Robotic Arm

To utilize the path finding algorithm in a practical manner, it was decided to integrate the algorithm with a robotic arm. The PUMA robot is a revolute robotic manipulator that has a work circle of approximately thirty-two inches in radius. It operates with its own programming language, VAL-II. The algorithm was translated from C into VAL and entered into the robots controller.

Using a board with a grid to simulate a shop floor, the optimal path was traced by nozzling sand from a reservoir positioned in the gripper of the robot. Hence, the robot's end manipulator simulated a single robot traversing the grid. The PUMA proved to be an impressing method of simulating the kinematic movements of the GKS simulation. Also evident was the fact that the path-finding algorithm can indeed be incorporated in a practical situation.

Theory

3.1 Determination of Paths

The total possible number of paths for N objects is N!, but after careful examination of the total distances it can be observed that there really is only N paths with many paths having equivalent distances. Figure 1 and Table 1 establishes all the possible paths that a robot may traverse in order to pick up three objects.

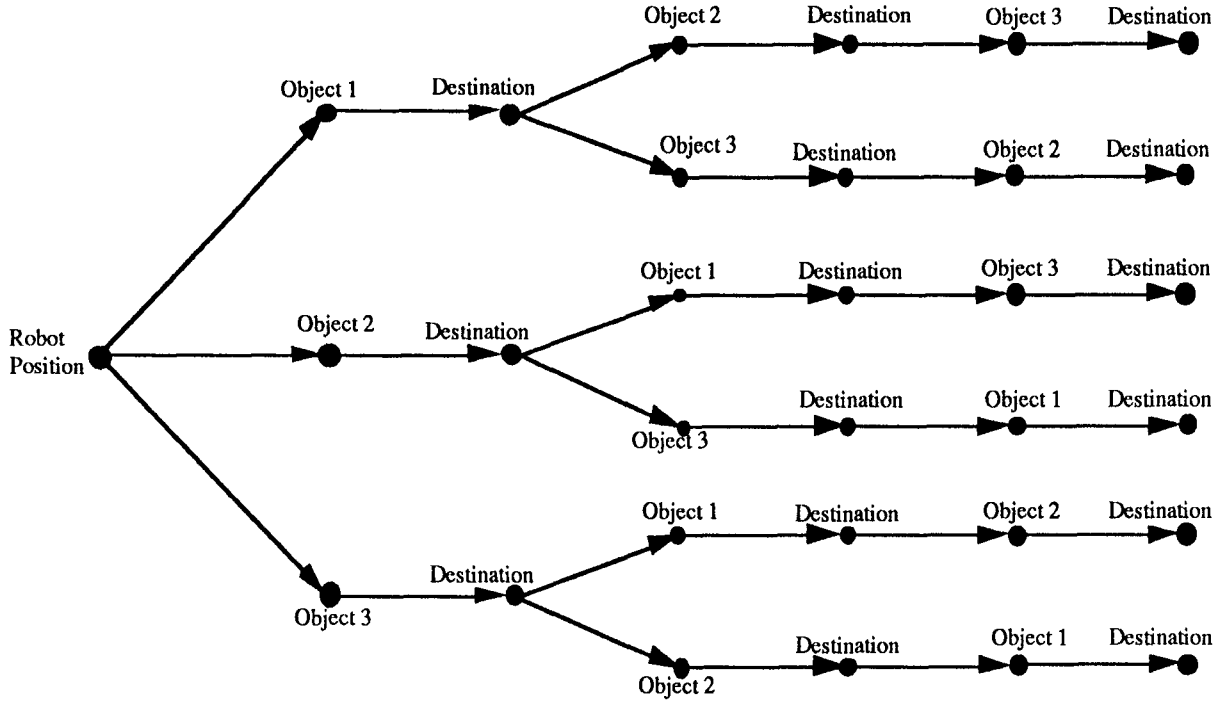


Figure 1: All possible paths a robot may traverse in order to retrieve 3 objects

Table 1: All possible paths a robot may traverse in order to retrieve 3 objects

Path	Segment 1	Segment 2	Segment 3	Segment 4	Segment 5	Segment 6	Order to Pick Objects
1	robot origin to object 1	object 1 to destination	destination to object 2	object 2 to destination	destination to object 3	object 3 to destination	123
2	robot origin to object 1	object 1 to destination	destination to object 3	object 3 to destination	destination to object 2	object 2 to destination	132
3	robot origin to object 2	object 2 to destination	destination to object 1	object 1 to destination	destination to object 3	object 3 to destination	213
4	robot origin to object 2	object 2 to destination	destination to object 3	object 3 to destination	destination to object 1	object 1 to destination	231
5	robot origin to object 3	object 3 to destination	destination to object 1	object 1 to destination	destination to object 2	object 2 to destination	312
6	robot origin to object 3	object 3 to destination	destination to object 2	object 2 to destination	destination to object 1	object 1 to destination	321

The total distance calculated for each object is determined from the simple distance equation:

$$distance_{12} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \text{(Eq. 1)}$$

If path 1 (123) and path 2 (132) are compared with each other it can be observed that in both cases the robot must travel from the destination to object 2, object 2 to the destination, the destination to object 3, and object 3 to the destination, not specifically in this order. In both cases the robot must first travel from its origin to object 1 to the destination. Since the first two segments the robot must travel are identical, the only variations are in the last four segments. If the last four segments are rearranged (Table 2 & Eq's 2-5) it can be seen that the total distances that must be traveled for these two cases are identical.

Table 2: Rearrangement of segments 4-6

Path	Segment 3	Segment 4	Segment 5	Segment 6	Order to Pick Objects
1	destination to object 2	object 2 to destination	destination to object 3	object 3 to destination	123
2	destination to object 3	object 3 to destination	destination to object 2	object 2 to destination	132

Path 1 (Segment 3) = Path 2 (Segment 5) (Eq. 2)

Path 1 (Segment 4) = Path 2 (Segment 6) (Eq. 3)

Path 1 (Segment 5) = Path 2 (Segment 3) (Eq. 4)

Path 1 (Segment 6) = Path 2 (Segment 4) (Eq. 5)

Therefore paths 3 through 6 are equivalent to each other and the there are only N number of possible distances for N! number of possible paths.

3.2 The Optimal Path-Finding Algorithm

After the coordinates of the objects, destination, and robot are obtained the total possible number of paths and total distances are calculated. The VAL-II program prompts the user while the C program reads an external file in order to obtain the necessary information. Once this information is obtained and the distances are determined, the total distances are sorted[2] in order to determine the shortest path. The sorted distances is the order in which the robot is to travel.

[2] The sorting routine was written and obtained from Dr. William Hruschka at the Instrumentation and Sensing Lab in Beltsville, Maryland. This sorting routine is a modified Quicksort which has been proven to be most optimal by a benchmark program. This sort divides an array by a factor of two in every iteration until finally every item is sorted. This is the unique part of this program because it eliminates much unnecessary reverse inspection.

3.3 The Graphics Kernal System Algorithm[3]

A grid is drawn on the screen with simple draw line statements, i.e. draw line from (x_1,y_1) to (x_2,y_2). A title, subtitle, axis labels, and axis values are draw by drawing strings to the screen. A robot is drawn represented by a triangle, a destination represented by a square, and the objects represented by circles. The position of every object on the grid is stored in separate arrays. The reason for this is the method in which the robot is shown to move. The robot is drawn in black on a white background. The robot is then redrawn in white, therefore erasing the robot from the screen, and a new robot is draw in black at a new position $(x_1+\partial x_1,y_1+\partial y_1)$. This drawing and redrawing bluffs the user into seeing the robot move across the screen. As each new robot is drawn the objects, destination, and grid lines that were erased must be redrawn, therefore the position of every object, destination, and grid line on the screen must be recorded. Each time the robot is redrawn the position of the robot is checked with the position of the grid lines, destination, and objects to determine if anything requires redrawing. The order in which the robot is to traverse is stored in an array which is then passed into a loop where the coordinates of the corresponding objects may be determined. The robot is then moved between the corresponding coordinates.

[3] Number_String algorithm was written and obtained form Michael A. Robbins at the Instrumentation and Sensing Lab in Beltsville, Maryland.

Apparatus

4.1 PUMA 560 Robot (Specifications)[4]

Manufacturer
Unimation, Incorporated
Shelter Rock Lane
Danbury, CT 06810
(203) 744-1800

Local Distributor
BHS Machinery Company
717 Airport Boulevard
South San Francisco, CA 94080
(415) 761-0131

Classification
Controlled Path (Point-to-Point)

Price Range
$41,000 to $80,000
(with Univision)

Drive Type
Electric DC Servo

Load Capacity
5.5 lbs. approx.

Repeatability
±0.004 in.

Reach
36 in.

Specifications

Positioning Performance
Repeatability
±0.004 in.

Manipulator
Drive Type
Electric Servo
Load Capacity
5.5 lbs. approx.
Configuration
Jointed Arm
Coordinate System
Joint; World (Cartesian),
Tool (Cartesian)
Degrees of Freedom
5 Axes
(PUMA 560: 6 Axes)
Gripper Actuation
Pneumatic

Power Requirement
110/130 VAC; 1Ø; 50/60 Hz
500 W

Controller
Type
Microcomputer
(Dec LSI-11 w/VAL language)
Programming Methods
Teach Pendant; CRT or TTY
Terminal (option)
Memory Capacity
16 K
External Storage
Floppy Disk (option)
I/O Interfaces
8 in/8 out; to 32 in/32 out
(option)
Cable Length
50 ft. max.

Environment
50 to 120°F
10 to 80% Relative Humidity

Special Notes/Options

Options
– CRT or TTY terminal
– Floppy disk memory storage
– 'Univision' system
– Additional I/O modules
(8 in/8 out each)
– Grippers
Alternate version PUMA 560 has an
an additional wrist motion.

Floor Space and Approximate Net Weight
Manipulator
16 in. Diameter Base
120 lbs.
Control Unit
18.9 w X 12.6 h X 20.1 d in.
80 lbs.
Power Unit
(Not a separate unit)

[4] Wolovick, William A., ROBOTICS: Basic Analysis and Design, Figure 1.3.5, The PUMA 550 (560) Specifications, CBS College Publishing, ©1987, pg. 23.

4.2 SUN Graphics Kernal System

The Graphics Kernal System (GKS) is a collection of precompiled algorithms created to assist the programmer create graphics on a SUN workstation screen. The lowest level of this library consists of turning on and off pixels. For example, a draw line command loops from x_1 to x_2 turning each individual pixel on. This package is very useful and easy for the programmer to interface a simulation with the computers' screen.

Procedures

- Practical Training
- Off-Line Programming
  - Input Module
  - Evaluation of Distance
  - Determination of Optimal Path
  - Animation
- Implementation on Industrial Robot

Discussion

6.1 Scope

Today, industrial robots are used in wide diversity of applications, such as welding, material handling, assembly, and spray-painting. Many future assembly operations will involve more than just a single, independently controlled robots, which perform specific tasks exclusive of their surroundings. More likely, appropriate groups of robots, along with associated peripheral equipment, such as conveyors, sensors, buffers, and feeders will be combined to form an entire manufacturing cell for the production of more complex products.

The objective of this experimentation is to in fact develop an algorithm which will enable a set of mobile robotic devices to simultaneously perform predetermined tasks without interference and to do so in an optimal way. This algorithm is a skeleton of a more sophisticated task-level robotic programming language which will enable robots to perform normal operations more efficiently and enable the controller to make appropriate decisions based on unpredictable and/or unknown circumstances.

In order to accomplish the task put before the group the project was broken down into four separate parts:

1. Practical Training
2. Off-Line Programming
3. Simulation
4. Implementation on a Robot Controller

6.2 Practical Training

Practical training included becoming familiar with the industrial robot. This training provided a much better appreciation and understanding robot uses in industry and the control of these robots.

6.3 Off-Line Programming

Off-Line Programming consisted of the development of a computer program, which consists of:

1. Input module
2. Evaluation of total transportation distance for N objects
3. Determination of the shortest path
4. Computer simulation using GKS (Graphic Kernal System)

6.3.1 Input Module

The input module enabled the user it enter in the robots initial position as well as the positions of N objects and the destination of these objects. This part of the program enabled the robot to be flexible and eligible to adapt to any configuration of objects.

6.3.2 Evaluation of Distance

With the coordinates of the objects known now lies the problem of the order of transportation of these objects to their destination. With N number of objects there are N! number of paths the robot may take. From geometry, knowing that the shortest distance between two points is a straight line the optimal path would then be the shortest distance traveled by the robot for our two dimensional planar case.

Knowing all the above parameters and using geometry, a simple solution to our problem is at hand. Let X'X, Y'Y, Z'Z (called the x-axis, the y-axis, and the z-axis, respectively) be three mutually perpendicular lines in space intersecting in a point O

(called the origin), forming in this way three mutually perpendicular planes XOY, XOZ. YOZ (called the xy-plane, the xz-plane, and the yz-plane, respectively). Then any point P of space is located by its signed distances x, y, z from the y-z plane, the xz-plane and the xy-plane, respectively, where x and y are the rectangular coordinates with respect to the axes X'X and Y'Y of the orthogonal projection P' of P or the xy-plane and z is taken as positive above and negative below the xy-plane and z is taken as positive above and negative below. The ordered triple of numbers, (x,y,z), are called rectangular coordinates of the point P.[5]

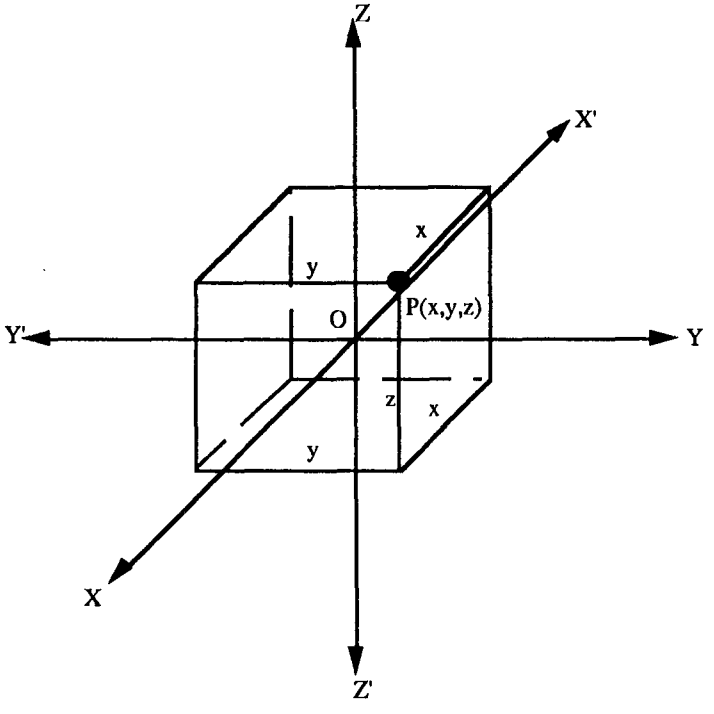


Figure 2: Rectangular Coordinates

Using the rectangular coordinate system, let P_1 (x_1, y_1, z_1) and P_2 (x_2, y_2, z_2) be any two points therefore, by analytic geometric theorems the distance between P_1 and P_2 is:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \qquad \text{(Eq. 6)}$$

[5] Beyer, W.H., CRC Standard Mathematical Tables, 27th Edition: CRC Press, Inc., ©1984.

In our planar case the equation further reduces to:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \text{(Eq. 7)}$$

6.3.3 Determination of Optimal Path

Using the above equation nested in a subroutine the total distance for all possible paths was obtained by first calculating the distance from the robot to the object, to the destination and also the distance from the destination to the object and back to the destination. Below is the subroutine in C programming language:

```
/*  a[i] - distance from robot to object to destination
    b[i] - distance from destination to object to destination
    x[i] - object i x coordinate
    y[i] - object i y coordinate
    rx - robot x coordinate
    ry - robot y coordinate
    dx - destination x coordinate
    dy - destination y coordinate       */

for (i=1; i<= N; i++)
{
        a[i] = pow((pow((x[i] - rx),2.0) + pow((y[i] - ry),2.0)),0.5);
        b[i] = pow((pow((dx - x[i],2.0)+pow((dy - y[i],2.0)),0.5);
        a[i] = a[i] + b[i]
        b[i] = 2 * b[i]
}
```

This subroutine coupled with a subroutine to determine the total distance traveled by the robot for every possible path was utilized. Below is the program which goes

through every possible path and calculates the total distance of all the paths which can be taken:

```
/*  determine the every possible path
    c[i] - is the total distance traveled if object i is picked  up first   */

for (i=1; i<=N; i++)
{
        c[i] = 0.0
        c[i] = a[i]
        for (j = 1; j <= N; j++)
        {
                if (i != j)
                        c[i] = c[i] + b[j];
        }
}
```

Using a Quick Sort routine which sorts the first and $N/2^{th}$ item, then the second and $N/2^{th}+1$ item, etc. After every item has been sorted it then divides the array by two and sorts the new fist item with the $N/4^{th}$ item. Each time the array is divided by two and sorted until every item is sorted. This method eliminates much unnecessary back checking and is one of the most optimal sorting routine.

6.3.4 Animation

Now, having all the calculation completed and sorted, the optimal path was easily recognized and animation or computer simulation was utilized to prove that the robot program developed is capable of:

1. Picking up the objects
2. Optimal time
3. Optimal motion that can coordinate with its surroundings

Animation allows a designer to see how things actually work without building a working prototype. It makes it possible to draw the individual components and then combine their images on the screen and to move them so as to examine their dynamic behavior. Any problems, that show up in this animation, can be solved by amending the design. The benefit of animation is to obviate the need for a working model and to greatly shorten the design time by efficiently providing optimum solutions to problems. This animation routine for our algorithm created a window and combined all the functions necessary to display the grid, print values, objects, destination, and move the robot through the shortest part to pick up these objects.

6.4 Implementation on Industrial Robot

Using the above equation nested in a subroutine the total distance for all possible paths. The final part of the project was to implement the algorithm on an actual robot. The original program, written in C programming language, was transformed into Val-II robot programming language and keyed into the Puma 560 controller. With the addition of a few simple move commands:

Moves shift (ab5 by xo[sx],yo[sy])
Moves shift (ab5 by xd,yd)

Allowed the robot to identify the shortest path and execute these moves by moving in a straight line from a known position, ab5, to the objects coordinates, xo and yo, and then to the destinations coordinates, xd and yd.

6.5 Future Research

Although the first phase of this project is complete the project as a whole is far from complete. Other options that need to be addressed are the following:

1. Three Dimensional Space
2. Multiple Mobile Robotic Devices
3. Detecting the collision of obstacles or other moving objects
4. Use of external sensors to aid in the optimization process.

6.5.1 Three Dimensional Space

Three Dimensional Space is important to consider because the configuration of the object to be picked up plays a key role in the time required for a robot to complete a task. Our algorithm assumes that all objects are symmetrical and the time required to pick them up is equivalent. In reality, no two objects are the same shape, therefore the angle of approach and orientation of the robot arm will play a key role in the reduction of time. It may in fact be more time effective and efficient for the robot to take a path with a longer distance so that it may approach a "hard to get to object" at the right orientation in order to pick it up in a minimal amount of time. In this particular case the shortest path is not always the optimal one.

6.5.2 Multiple Mobile Robots

Multiple mobile robots also introduce many other problems in the optimization of time. Mobile robotic arms must be placed on carts or wheels to enable steering and locomotion. This will restrict the ability of the robot to change directions instantaneously and also it must be able to account for the motion it undergoes as it turns around a corner. One solution to this is to design a robot with minimum turning radius, because the turning radius along with deceleration in the turns adds complexity to the optimal path algorithm.

6.5.3 Obstacles

Another problem is the detection of obstacles and collision with them. The algorithm must be able to plan the robots path and predict the path of other moving object so as not to interfere and cause a collision. This is a very difficult hurdle to overcome because not only is it a static problem but also a dynamic problem. Often a robot has no control of the behavior of the objects and the surrounding environment may contribute other factors in the interaction.

6.5.4 External Sensors

However, by the introduction of sensors combined with the optimal path algorithm we are moving in the direction of granting the robot with the authority to make decisions. These sensors will allow the robot to perform more sophisticated tasks such as, catching, tracking, and moving around objects.

Conclusion

The field of robotics and computerized automation is a dynamic one with developments proceeding at an extremely fast pace. The leading edge of research aims to find solutions to industrial problems based upon existing technology. The optimal path algorithm is an example of developmental research that may some day benefit the industrial community. The main accomplishments of this research was:

1. Evaluation of total transportation distance
2. Determination of the shortest path
3. Computer animation to prove validity of shortest path
4. Implementation of algorithm on Puma 560 industrial robot

Acknowledgements

Special thanks to Professor Jackson Yang and his research group for letting up use his robotics facilities and to Dr. William Hruschka and Michael Robbins at the Instrumentation and Sensing Laboratory in Beltsville, Maryland for helping us with certain aspects of the computer algorithm.

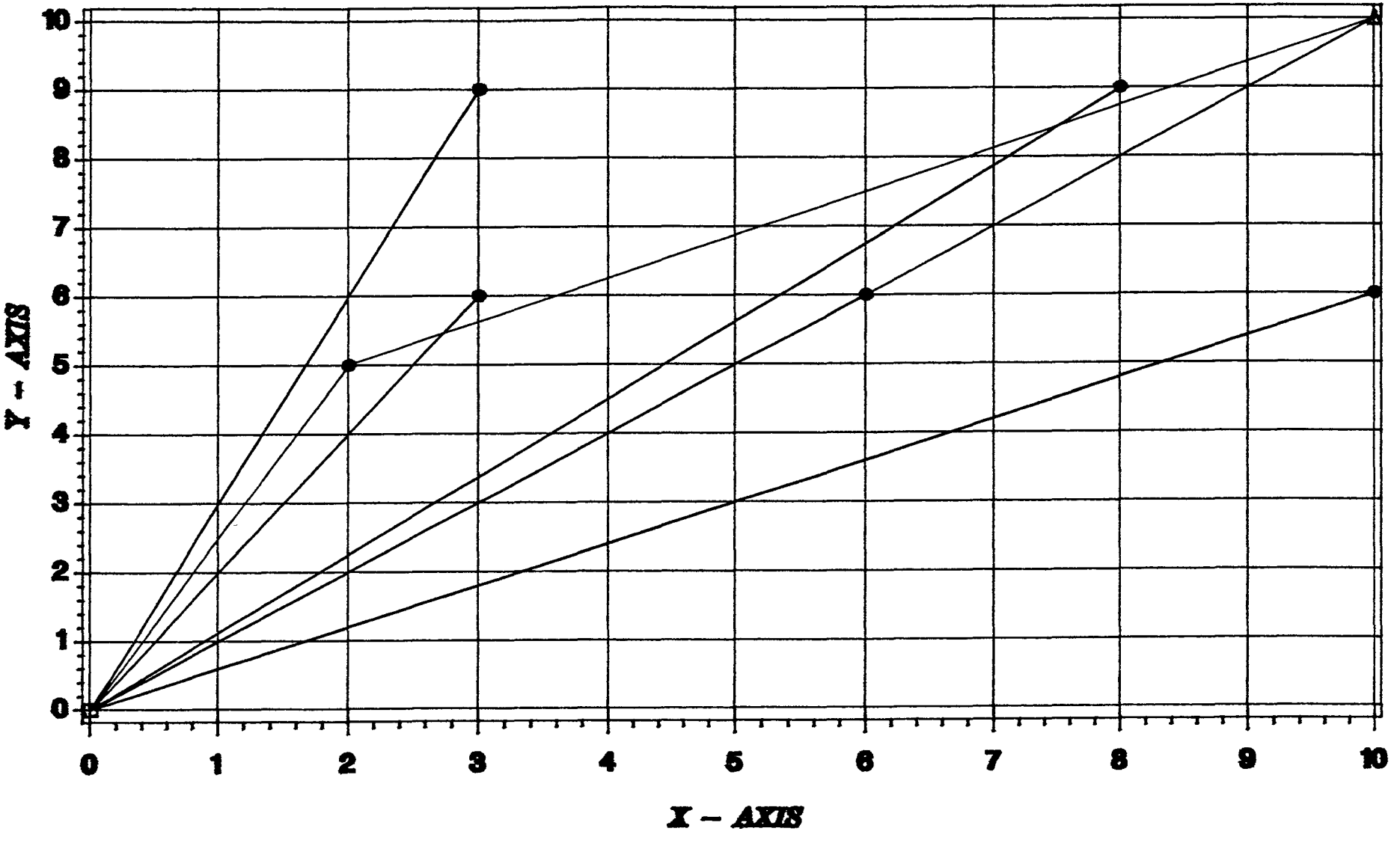
References

1. Grover, M.P., Automation, Production Systems, and Computer - Integrated Manufacturing, Part IV: Industrial Robots, Prentice Hall, ©1987.
2. Wolovich, William A., ROBOTICS: Basic Analysis and Design, CBS College Publishing, ©1987.
3. D. Sharon, J. Harstein, G. Yartion, Robotics and Automated Manufacturing, Pitman Publishing, ©1987.
4. Billingsly, J., Robotics and Automated Manufacture, Peter Peregrinus, Ltd., ©1988.
5. Beyer, W.H., CRC Standard Mathematical Handbook, 27th Edition, CRC Press, Inc., ©1984.
6. P. Jacobs and J. Canny, Planning Smooth Paths for Mobile Robots, IEEE Computer Society Press, ©1989.
7. Gilbert E.G. and Hong S.M., A New Algorithm for Detecting the Collision of Moving Objects, IEEE Computer Society Press, ©1989.

Appendix A: GKS Simulation Screen

Optimal Path Finding Algorithm for Mobile Robotic Devices

Robotic Animation



Appendix B: Optimal Path-Finding Algorithm Integrated with GKS Software

```
#include <stdio.h>
#include <math.h>
#include <gks/ansicgks.h>

/*
 * This program was written by:
 *
 *                              Thomas Lu
 *                              Michael O'Malley
 *                              Bret Tegeler
 *                              Craig Baker
 *
 * for ENME 480, a project class supervised by Professor Guangming Zhang.
 * This program reads the coordinates of a robot, a destination, and N
 * number of objects from the file "COORDINATES".  It then determines all
 * possible paths the robot may take and sorts for the shortest path.
 * The route the robot is to take is then animated with the help of the
 * GKS (graphics kernal system) library on the SUNOS system.  This library
 * is only in the SRC-computer laboratory and will only run there.
 *
 * To compile this program either type:
 *
 *      cc 480.c -o 480 -g -lgks -lsuntool -lsunwindow -lpixrect -lm
 *
 * or create a file named 'go'.  Within this type
 *
 *      cc $1.c -o $1 -g -lgks -lsuntool -lsunwindow -lpixrect -lm
 *
 * then make the file executable by typing "chmod +x go".  So when you want
 * to compile 480.c you can type go 480.  This will save you the time
 * in typing the long line and replaces $1 in go with 480.  This also is
 * a general compiling routine that will allow you to compile any *.c
 * program by typing go [file]
 *
 * To make a hard cory of the text output just type '480 > name of file'
 * This will (>) 'redirect' the output to the file you have named
 *
 */

#define MAX(a,b) (a<b?b:a)
#define MIN(a,b) (a<b?a:b)

/*
 *
 * Define Statement:
 *
 *      #define MAX(a,b) (a<b?b:a)
 *
 *      ? = then
 *      : = else
 *
 *      if (a < b) then
 *              return(a)
 *      else
 *              return(b)
 *
 */

double pow();
double Translate();

double  rx,ry;
double  dx,dy;
int     N;
double  x[200],y[200];
double  a[200],b[200];
```

```
double  c[200],d[200];
int     i;
int     order[200];
double  xmin,xmax;
double  ymin,ymax;

Gnpoint point;
Gnpoint shift;
int inc = 20;
Gwpoint robot[5];
Gwpoint object[200];
Gwpoint destination;
Gwpoint line[200];

/*                      Global Variables & Functions
 *
 * Functions:
 *      pow()           pow(X,Y) = X raised to the Y(th) power
 *      Translate()     converts our coordinates to window coord (ndc-dc)
 *
 * Variables:
 *      rx,ry           coordinates of the robot
 *      dx,dy           coordinates of the destination
 *      N               number of objects
 *      x[i],y[i]       coordinates of the objects
 *      a[i]            distance from robot to object to destination
 *      b[i]            distance from destination to object to destination
 *      c[i]            total distance traveled by the robot
 *      d[i]            copy of c[i] so that the path may be determined
 *      i               arbitrary increment
 *      order[i]        the order of objects the robot should pick up
 *      xmin,xmax       min and max in the x axis
 *      ymin,ymax       min and max in the y axis
 *
 *      point           position of the robot as it is moving
 *      shift           how much to shift between each inc
 *      inc             number of times robot drawn between 2 distances
 *      robot[5]        dimensions of the robot
 *      object[200]     window coordinates of the objects
 *      destination     window coordinates of the destination
 *      line[200]       window coordinates of the grid
 *
 */

main()
{
        GetCoordinates();
        GetDistances();
        GetPaths();
        QuickSort();
        MinMax();
        Animate();
}


/*
 *
 *                      GetCoordinates()
 *
 *      Opens the file "COORDINATES" and reads the robot coordinates,
 * destination coordinates, number of objects, and objects coordinates.
 * Also determines if any of the coordinates are the same and exits if
 * they are.
 *
 * Functions:
 *      fopen()         opens the file *filename and checks if accessable
```

```
 *                              "r" stands for read only
 *      fgets()         reads a character at a time up to 100 characters
 *                              or until end of line from "file" and places
 *                              the string into "name"
 *      sscanf()        converts the string "name" of %lf (long float) type
 *                              to a number in array "value[i]"
 *
 * Variables:
 *      filename        name of file coordinates are held in
 *      file            accessability of the file
 *      j               checks for EOF
 *      name            string value of coordinates
 *      value[i]        numberic value of coordinates
 *      k               arbitrary increment
 *
 */

GetCoordinates()
{
        char name[100];
        char j = 2;
        FILE *file;
        char *filename = "COORDINATES";
        double value[200];
        int k;

        /* open file as read only */
        file = fopen(filename,"r");
        if (file == 0)
        {
                printf("Can't open file COORDINATES \n");
                exit(0);
        }

        /* while not EOF read in data */
        while (j >= 1)
        {
                j = fgets(name,100,file);
                if (j != 0)
                {
                        sscanf(name,"%lf",&value[i]);
                        i++;
                }
        }

        rx = value[0];
        ry = value[1];
        dx = value[2];
        dy = value[3];
        N  = (int) value[4];

        j = 5;
        for (i = 1; i <= N; i++,j+=2)
        {
                x[i] = value[j];
                y[i] = value[j + 1];
        }

        /* checks to detrmine if any of the coordinates are the same */
        /* robot & destination */
        if (rx == dx && ry == dy)
        {
                printf("Several of the coordinates are equivalent.\n");
                printf("Please check the file COORDINATES.\n");
                exit(0);
        }
```

```
        /* robot,destination & objects */
        for (i = 1; i <= N; i++)
        {
                if (rx == x[i] && ry == y[i] || dx == x[i] && dy == y[i])
                {
                        printf("Several of the coordinates are equivalent.\n");
                        printf("Please check the file COORDINATES.\n");
                        exit(0);
                }
        }

        /* object[i] & object[j] */
        for (i = 1; i <= N; i++)
        {
                for (k = 1; k <= N; k++)
                {
                        if (i != k)
                        {
                                if (x[i] == x[k] && y[i] == y[k])
                                {
                        printf("Several of the coordinates are equivalent.\n");
                        printf("Please check the file COORDINATES.\n");
                        exit(0);
                                }
                        }
                }
        }

        printf("\n");
        printf("Robot's Coordinates              (%4.1lf,%4.1lf) \n",rx,ry);
        printf("Destination's Coordinates        (%4.1lf,%4.1lf) \n",dx,dy);
        printf("Number of Objects                %4.2d \n",N);
        printf("\n");
        printf("\t Object \t x[i] \t y[i] \n");

        for (i = 1; i <= N; i++)
                printf("\t   %d \t\t %4.2lf \t  %4.2lf \n",i,x[i],y[i]);
}


/*
 *                      GetDistances()
 *
 *      Calculates the distances from the robot to objects to the
 * destination {a[i]} and also the distance from the destination to the
 * object and back to the destination {b[i]}.
 *
 */

GetDistances()
{
        int j;

        printf("\n\n\n\n");
        printf("Distances Between Robot, Objects, and Destination \n\n");
        printf("\t  a[i] \t\t  b[i] \n");

        /* a[i] - distance from robot to object to destination
           b[i] - distance from destination to object to destination */
        for (i = 1;i <= N;i++)
        {
                a[i] = pow((pow((x[i] - rx),2.0) + pow((y[i] - ry),2.0)),0.5);
                b[i] = pow((pow((dx - x[i]),2.0) + pow((dy - y[i]),2.0)),0.5);
                a[i] = a[i] + b[i];
                b[i] = 2 * b[i];
```

```
                printf("\t %6.4lf \t %6.4lf \n",a[i],b[i]);
        }
}


/*
 *                      GetPaths()
 *
 *      Calculates all the possible paths and the total distances the
 * robot takes and places the distances into the array {c[i]}.
 *
 */

GetPaths()
{
        int j;

        /* determine the every possible path */
        for (i = 1; i <= N; i++)
        {
                c[i] = 0.0;
                c[i] = a[i];
                for (j = 1; j <= N; j++)
                {
                        if (i != j)
                                c[i] = c[i] + b[j];
                }
        }

        /* if 2 paths are equal then add epsilon to one of them */
        for (i = 1; i <= N; i++)
        {
                for (j = 1; j <= N; j++)
                {
                        if (i != j && c[i] == c[j])
                                c[i] = c[i] + 0.00001;
                }
        }

        printf("\n\n\n\n");
        printf("Total Distances \n\n");
        printf("\t Object \t   c[i] \n");

        for (i = 1; i <= N; i++)
                printf("\t   %d \t\t %6.4lf \n",i,c[i]);
}

/*
 *                      QuickSort()
 *
 *      This sorting routing sorts the 1st and N/2(th) item, then the
 * 2nd and N/2(th)+1 item, etc.  After every item has been sorted it then
 * divides the array by 2 and sorts the new 1st item with the N/4(th) item.
 * Each time the array is divided by 2 and sorted until nrec = 1 and every
 * item is sorted.  This method eliminates much unnecessary back checking
 * and is one of the most optimal sorting routines. Stores the order in
 * which to pick the objects up.
 *
 * Variables:
 *      j,k,l,m         arbitrary counter
 *      it              temporary storage for switching values
 *      nrec            divides the array by 0.5 each time
 *      order[]         order in which to pick up objects
 *
 * Acknowledgement:  This program originally written by
 *
```

```
 *              Dr. William Hruschka
 *              USDA/ARS/PQDI
 *              Instrumentation and Sensing lab
 *              Beltsville, Maryland 20705
 *              (301) 344-3650
 *
 */

QuickSort()
{
        int nrec,j,k,l,m;
        double it;
        double total = 0.0;

        /* make copy of distances */
        for (i = 1; i <= N; i++)
                d[i] = c[i];

        /* Quicksort */
        nrec = N;
        m = nrec/2;
        while (m > 0)
        {
                l = nrec - m;
                k = 1;
                while (k <= l)
                {
                        i = k;
                        j = 0;
                        while ((i > 0) && (j == 0))
                        {
                                j = i + m;
                                if (d[i] > d[j])
                                {
                                        it = d[i];
                                        d[i] = d[j];
                                        d[j] = it;
                                }
                                i = i - m;
                                j = 0;
                        }
                        k++;
                }
                m = m/2;
        }

        /* store the order to be taken by the robot in order[] */
        for (i = 1; i <= N; i++)
        {
                for (j = 1; j <= N; j++)
                {
                        if (d[i] == c[j])
                                order[i] = j;
                }
        }

        printf("\n\n\n\n");
        printf("Path To Take \n\n");
        printf("\t Order \t Object \t   Coordinates \t\t Distance \n");

        for (i = 1; i <= N; i++)
        {
                if (i == 1)
                {
                        printf("\t   %d \t   %d \t\t (%6.2lf,%6.2lf) \t %6.4lf \n",i,order
                        total = total + a[order[i]];
```

```
                }
                else
                {
                        printf("\t   %d \t   %d \t\t (%6.2lf,%6.2lf) \t %6.4lf \n",i,order
                        total = total + b[order[i]];
                }
        }

        printf("\n");
        printf("\t\t\t\t Total Distance          %6.4lf \n",total);
}


/*
 *                      MinMax()
 *
 *      Calculates the minimum and maximum x and y values.  These
 * values will be used later in order to draw the grid on the screen.
 *
 */

MinMax()
{
        xmin = ymin = 100000.0;
        xmax = ymax = 0.0;

        /* Calculate min/max of objects */
        for (i = 1; i <= N; i++)
        {
                xmin = MIN(x[i],xmin);
                xmax = MAX(x[i],xmax);
                ymin = MIN(y[i],ymin);
                ymax = MAX(y[i],ymax);
        }

        /* Include destination in calculation of min & max */
        xmin = MIN(dx,xmin);
        xmax = MAX(dx,xmax);
        ymin = MIN(dy,ymin);
        ymax = MAX(dy,ymax);

        /* Include robot in calculation of min & max */
        xmin = MIN(rx,xmin);
        xmax = MAX(rx,xmax);
        ymin = MIN(ry,ymin);
        ymax = MAX(ry,ymax);
}

/*
 *                      Animate()
 *
 *      This routine creates a window and combines all of the functions
 * necessary to display the grid, print values, display objects, display the
 * destination, and move the robot.
 *
 */

Animate()
{
        Gchar *conn = NULL;
        Gchar *wstype = "sun_tool";
        Gws ws = 1;

        /* Create and Open Window */
        if (gopengks(stdout,GMEMORY))
        {
```

```
                printf("Unable to open window \n");
                exit(2);
        }
        gopenws(ws,conn,wstype);
        gactivatews(ws);

        gsetfillcolour(1);

        DrawGrid();
        DrawObjects();
        DrawStrings();
        MoveRobot();

        /* screendump to file sdump for printing */
        /* type 'screenload sdump' to see raster file
         * WARNING!!!! Don't try to print this file
         * It will print but is so large that it will
         * probably tie up the printer for an hour
         * or crash the printer spooler */
        system("screendump sdump");
        sleep(4);

        /* Unmanage and Destroy Window */
        gdeactivatews(ws);
        gclosews(ws);
        gclosegks();
}

DrawGrid()
{
        Gwpoint gdppts[6];

        /* draw vertical grid lines */
        for (i = (int) xmin; i <= (int) xmax; i++)
        {
                /* store the coordinates so they may be redrawn later */
                line[i].x = Translate((double) i,xmin,xmax);
                gdppts[0].x = Translate((double) i,xmin,xmax);
                gdppts[0].y = 0.1;
                gdppts[1].x = gdppts[0].x;
                gdppts[1].y = 0.9;

                gpolyline(2,gdppts);
        }

        /* draw horizontal grid lines */
        for (i = (int) ymin; i <= (int) ymax; i++)
        {
                /* store the coordinates so they may be redrawn later */
                line[i].y = Translate((double) i,ymin,ymax);
                gdppts[0].x = 0.1;
                gdppts[0].y = Translate((double) i,ymin,ymax);
                gdppts[1].x = 0.9;
                gdppts[1].y = gdppts[0].y;

                gpolyline(2,gdppts);
        }
}

DrawObjects()
{
        Gwpoint gdppts[6];
        Ggdpi circgdp();
        Ggdprec datrec;
        Gflinter intstyle = SOLID;
```

```
        /* draw each object */
        for (i = 1; i <= N; i++)
        {
                gdppts[0].x = Translate(x[i],xmin,xmax);
                gdppts[0].y = Translate(y[i],ymin,ymax);
                gdppts[1].x = gdppts[0].x;
                gdppts[1].y = gdppts[0].y + 0.01;

                ggdp(2,gdppts,gcirgdp,&datrec);

                /* store the coordinates of each object */
                object[i].x = gdppts[0].x;
                object[i].y = gdppts[0].y;
        }

        gsetfillintstyle(intstyle);

        /* store the coordinates of the destination */
        destination.x = Translate(dx,xmin,xmax);
        destination.y = Translate(dy,ymin,ymax);

        /* draw the destination */
        gdppts[0].x = destination.x + 0.01;
        gdppts[0].y = destination.y + 0.01;
        gdppts[1].x = gdppts[0].x - 0.02;
        gdppts[1].y = gdppts[0].y;
        gdppts[2].x = gdppts[0].x - 0.02;
        gdppts[2].y = gdppts[0].y - 0.02;
        gdppts[3].x = gdppts[0].x;
        gdppts[3].y = gdppts[0].y - 0.02;
        gdppts[4].x = gdppts[0].x;
        gdppts[4].y = gdppts[0].y;

        gfillarea(5,gdppts);
}

MoveRobot()
{
        Gflinter intstyle = SOLID;
        gsetfillintstyle(intstyle);

        /* coordinate of the robot as it is moving */
        point.x = rx;
        point.y = ry;

        /* draw robot at origin */
        Robot(1);
        gfillarea(4,robot);
        sleep(2);

        /* move robot to first object */
        Move(rx,ry,x[order[1]],y[order[1]]);
        /* erase first object to be picked up */
        Erase(order[1]);

        /* move from object to destination to the next object */
        for (i = 1; i < N; i++)
        {
                Move(x[order[i]],y[order[i]],dx,dy);
                Move(dx,dy,x[order[i + 1]],y[order[i + 1]]);
                /* erase each object as it is picked up */
                Erase(order[i + 1]);
        }
        /* move from the last object to the destination */
        Move(x[order[N]],y[order[N]],dx,dy);
        /* move from the destination back to robot origin */
```

```
        Move(dx,dy,rx,ry);
}

/*
 *                      Translate()
 *
 *      This routine converts ndc coordinates (the window coordinates)
 * to dc coordinates (my coordinates).
 *
 */

double Translate(val,min,max)
        double val;
        double min;
        double max;
{
        double nval;

        nval = (((val - min)/(max - min))*(0.9 - 0.1)) + 0.1;
        return(nval);
}

DrawStrings()
{
        Gchar *text;
        Gwpoint place;

        /* draw the title */
        place.x = 0.25;
        place.y = 0.97;
        text = "OPTIMAL PATH-FINDING ALGORITHM FOR MOBILE ROBOTIC DEVICES";
        gtext(&place,text);

        /* draw the subtitle */
        place.x = 0.40;
        place.y = 0.95;
        text = "ROBOTIC ANIMATION";
        gtext(&place,text);

        /* draw the y-axis label */
        place.x = 0.045;
        place.y = 0.92;
        text = "Y - AXIS";
        gtext(&place,text);

        /* draw the x-axis label */
        place.x = 0.87;
        place.y = 0.06;
        text = "X - AXIS";
        gtext(&place,text);

        /* draw the numbers of the x axis */
        for (i = ((int) ymin); i <= ((int) ymax); i+=2)
        {
                *text = '\0';
                place.x = 0.006;
                place.y = Translate((double) i, ymin,ymax) - 0.005;
                number_string(i,text,1);
                gtext(&place,text);
        }

        /* draw the numbers of the y axis */
        for (i = ((int) xmin); i <= ((int) xmax); i+=2)
        {
                *text = '\0';
                place.x = Translate((double) i, xmin,xmax) - 0.075;
```

```
                place.y = 0.075;
                number_string(i,text,1);
                gtext(&place,text);
        }

}

/*
 * number_string - converts a set of integer numbers into their
 *      equivalent representation in characters.
 *
 * Acknowledgement:  This routine originally written by
 *
 *              Michael A. Robbins
 *              USDA/ARS/PQDI
 *              Instrumentation and Sensing Lab
 *              Beltsville, Maryland 20705
 *              (301) 344-3650
 *
 */

number_string(fpoint,pstring,count)
        int fpoint;
        char *pstring;
        int count;
{
        int j,k;
        int integer;
        int charnum;
        char charary[16];
        char buf[8];

        if (fpoint != 0)
        {
                for (j = 0; j < count; j++)
                {
                        integer = abs(fpoint);
                        charary[15] = '\0';
                        for (k = 14; (integer != 0); k--)
                        {
                                charnum = integer % 10;
                                charary[k] = 0x30 + charnum;
                                integer /= 10;
                        }

                        if (fpoint < 0)
                                charary[k--] = '-';

                        for (; k > 6; k--)
                                charary[k] = ' ';

                        fpoint++;
                        strcat(pstring,&charary[7]);
                        if (j != (count - 1))
                                strcat(pstring,",");
                }
        }
        else
        {
                sprintf(buf,"       0");
                strcpy(pstring,buf);
        }

        return;
}
```

```
Move(x1,y1,x2,y2)
        double x1,x2,y1,y2;
{
        int j;
        Gwpoint gdppts[3];

        /* determine how many points to draw between the two objects */
        shift.x = (x2 - x1)/inc;
        shift.y = (y2 - y1)/inc;

        for ( j = 1; j <= inc; j++)
        {
                /* draw old robot in same color as background */
                gsetfillcolour(0);
                Robot();
                gfillarea(4,robot);

                /* set the forground to a color that can be seen */
                gsetfillcolour(1);

                /* draw the path of the robot */
                gdppts[0].x = Translate(x1,xmin,xmax);
                gdppts[0].y = Translate(y1,ymin,ymax);
                gdppts[1].x = Translate(point.x,xmin,xmax);
                gdppts[1].y = Translate(point.y,ymin,ymax);
                gpolyline(2,gdppts);

                /* redraw anything that has been re-exposed */
                Redisplay(robot[0].x + 0.01,robot[0].y + 0.01);

                /* increment and draw next position of the robot */
                point.x = point.x + shift.x;
                point.y = point.y + shift.y;
                Robot();
                gfillarea(4,robot);
        }
}

Robot()
{
        double px,py;

        /* this routine takes the current coordinates of the robot and
           converts them to the coordinates of the screen, then creates
           the coordinates of the robot */

        px = Translate(point.x,xmin,xmax);
        py = Translate(point.y,ymin,ymax);

        robot[0].x = px - 0.01;
        robot[0].y = py - 0.01;
        robot[1].x = robot[0].x + 0.02;
        robot[1].y = robot[0].y;
        robot[2].x = robot[0].x + 0.01;
        robot[2].y = robot[0].y + 0.02;
        robot[3].x = robot[0].x;
        robot[3].y = robot[0].y;
}

Redisplay(x,y)
        double x,y;
{
        Gwpoint gdppts[6];
        int j;
        Ggdpi circgdp();
        Ggdprec datrec;
```

```
        /* redisplay vertical lines */
        for ( j = (int) xmin; j <= (int) xmax; j++)
        {
                if (line[j].x >= (x - 0.01) && line[j].x <= (x + 0.01))
                {
                        gdppts[0].x = line[j].x;
                        gdppts[0].y = 0.1;
                        gdppts[1].x = line[j].x;
                        gdppts[1].y = 0.9;

                        gpolyline(2,gdppts);
                }
        }

        /* redisplay horizontal lines */
        for ( j = (int) ymin; j <= (int) ymax; j++)
        {
                if (line[j].y >= (y - 0.01) && line[j].y <= (y +  0.01))
                {
                        gdppts[0].x = 0.1;
                        gdppts[0].y = line[j].y;
                        gdppts[1].x = 0.9;
                        gdppts[1].y = line[j].y;

                        gpolyline(2,gdppts);
                }
        }

        /* redisplay objects */
        for (j = 1; j <= N; j++)
        {
                if (object[j].y >= (y - 0.01) && object[j].y <= (y + 0.01) &&
                        object[j].x >= (x - 0.01) && object[j].x <= (x + 0.01))
                {
                        gdppts[0].x = object[j].x;
                        gdppts[0].y = object[j].y;
                        gdppts[1].x = object[j].x;
                        gdppts[1].y = object[j].y + 0.01;

                        ggdp(2,gdppts,gcirgdp,&datrec);
                }
        }

        /* redisplay destination */
        if (destination.x >= (x - 0.02) && destination.x <= (x + 0.02))
        {
        if (destination.y >= (y - 0.02) && destination.y <= (y + 0.02))
        {
                gdppts[0].x = destination.x + 0.01;
                gdppts[0].y = destination.y + 0.01;
                gdppts[1].x = gdppts[0].x - 0.02;
                gdppts[1].y = gdppts[0].y;
                gdppts[2].x = gdppts[0].x - 0.02;
                gdppts[2].y = gdppts[0].y - 0.02;
                gdppts[3].x = gdppts[0].x;
                gdppts[3].y = gdppts[0].y - 0.02;
                gdppts[4].x = gdppts[0].x;
                gdppts[4].y = gdppts[0].y;

                gsetfillcolour(1);
                gfillarea(5,gdppts);
        }
        }
}
```

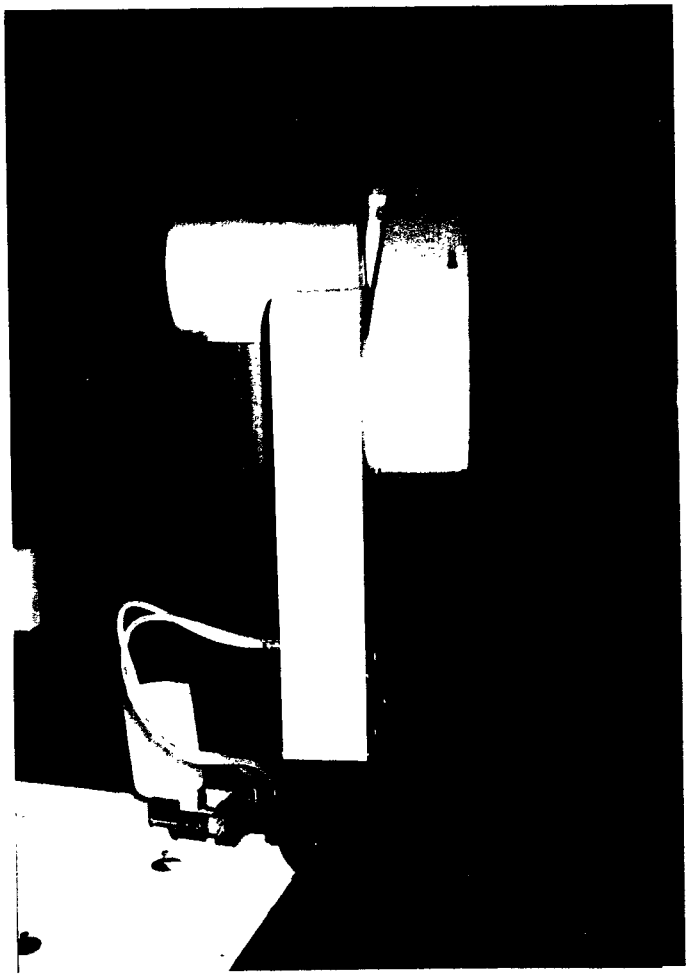
```
Erase(obj)
        int obj;
{
        Gwpoint gdppts[6];
        double xx,yy;

        /* if the object has been picked up, draw the object in the same
           color as the background, therefore erasing it.  Redraw anything
           that has been exposed, and set the new coordinates of the object
           to a coordinate that is not accessable to the robot */

        /* set the color to the same as the background */
        gsetfillcolour(0);

        /* erase object obj */
        xx = object[obj].x;
        yy = object[obj].y;

        gdppts[0].x = xx + 0.0125;
        gdppts[0].y = yy + 0.0125;
        gdppts[1].x = xx - 0.0125;
        gdppts[1].y = yy + 0.0125;
        gdppts[2].x = xx - 0.0125;
        gdppts[2].y = yy - 0.0125;
        gdppts[3].x = xx + 0.0125;
        gdppts[3].y = yy - 0.0125;
        gdppts[4].x = xx + 0.0125;
        gdppts[4].y = yy + 0.0125;

        gfillarea(5,gdppts);

        /* reset the foreground color */
        gsetfillcolour(1);

        /* reset the coordinates of the object to (0.00,0.00).  The robot
           can never reach this position because the window coordinates
           for the grid are from 0.1 through 0.9. */

        object[obj].x = 0.00;
        object[obj].y = 0.00;

        /* redisplay anything that has been exposed */
        Redisplay(gdppts[0].x - 0.0125,gdppts[0].y - 0.0125);
}
```

Appendix C: Optimal Path-Finding Algorithm Integrated with PUMA Robot

```
.P  GRAM zhang
    PROMPT "ENTER NUMBER OF OBJECTS", n
    PROMPT "ENTER ROBOT X COORDINATE", xr
    PROMPT "ENTER ROBOT Y COORDINATE", xy
    PROMPT "ENTER DESTINATION X COORDINATE", xd
    PROMPT "ENTER DESTINATION Y COORDINATE", yd
    FOR x = 1 TO n STEP 1
        PROMPT "ENTER OBJECT X COORDINATE", xo[x]
        PROMPT "ENTER OBJECT Y COORDINATE", yo[x]
    END
    FOR x = 1 TO n STEP 1
        a[x] = SQRT(SQR(xo[x]-xr)+SQR(yo[x]-xy))
        b[x] = SQRT(SQR(xo[x]-xd)+SQR(yo[x]-yd))
        a[x] = a[x]+b[x]
        b[x] = 2*b[x]
    END
    FOR i = 1 TO n STEP 1
        c[i] = 0
        c[i] = a[i]
        FOR j = 1 TO n STEP 1
            IF i <> j THEN
                c[i] = c[i]+b[j]
            ELSE
            END
        END
    END
    FOR i = 1 TO n STEP 1
        d[i] = c[i]
    END
    nrec = n
    m = INT(nrec/2)
    WHILE m > 0 DO
        l = nrec-m
        k = 1
        WHILE k <= l DO
            i = k
            j = 0
            WHILE ((i > 0) AND (j == 0)) DO
                j = i+m
                IF d[i] > d[j] THEN
                    it = d[i]
                    d[i] = d[j]
                    d[j] = it
                    i = i-m
                    j = 0
                ELSE
                    i = i-m
                    j = 0
                END
                k = k+1
            END
```

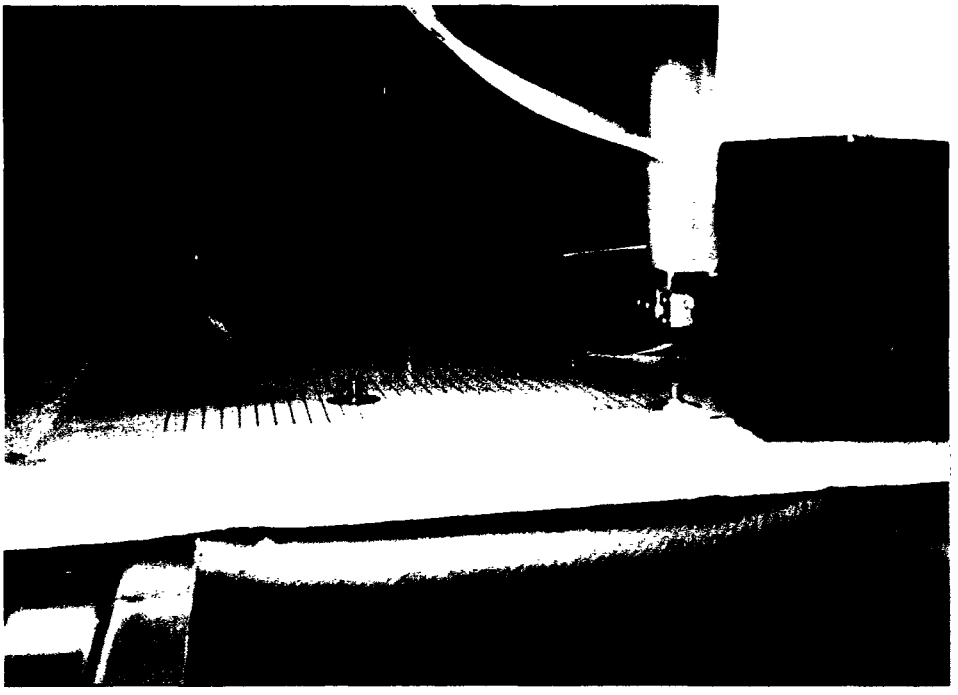
```
            n= INT(n/2)
        END
    END
    FOR i = 1 TO n STEP 1
        FOR j = 1 TO n STEP 1
            IF d[i] == c[j] THEN
                order[i] = j
            ELSE
            END
        END
    END
    FOR f = 1 TO n STEP 1
        sx = order[f]
        sy = order[f]
        MOVES SHIFT(ab5 BY xo[sx], yo[sy])
        DELAY 1
        MOVES SHIFT(ab5 BY xd, yd)
        DELAY 1
    END
    DELAY 1
    MOVES ab5
END
```

Appendix D: PUMA 560 Robot and Support Hardware

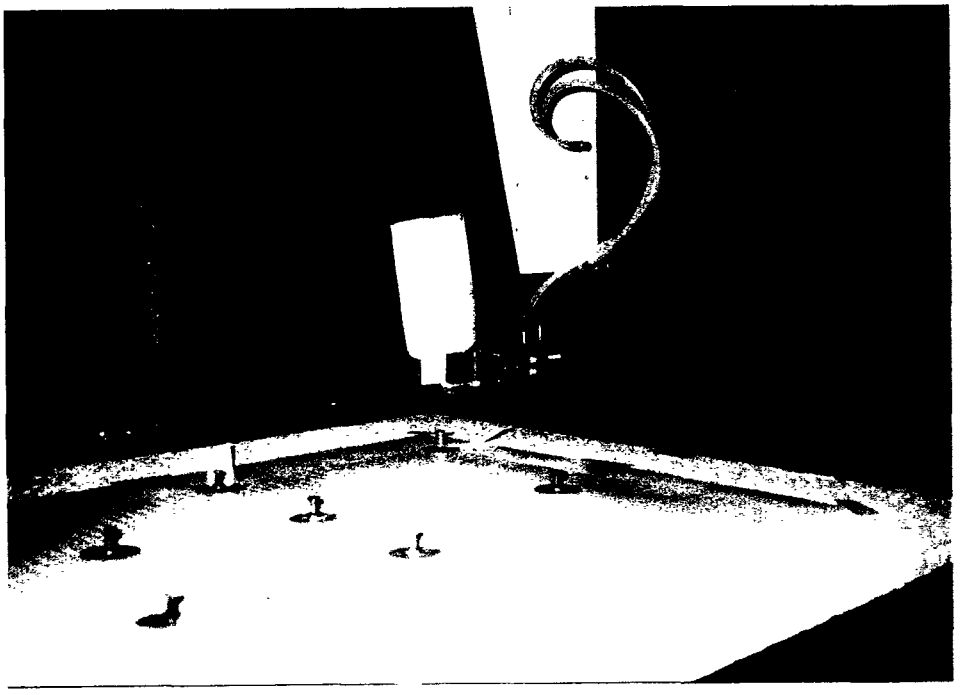


The Puma 560 industrial robot. Used in a wide diversity of applications such as welding, material handling, and assembly.

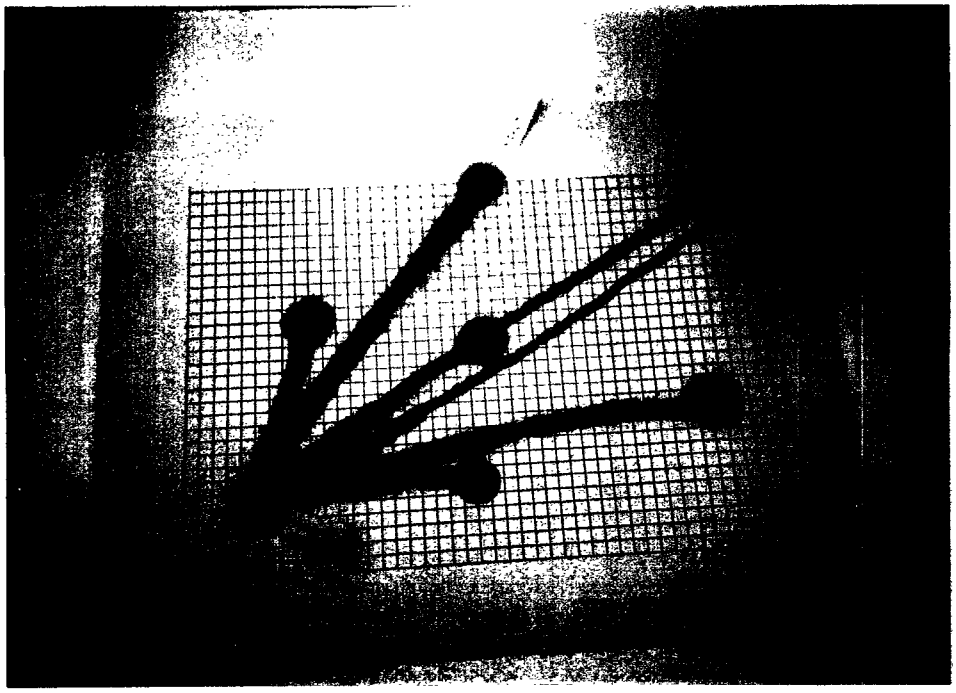
Entering the otimal path algorithm into the Unimation Controller in VAL-II, robotic programming language.



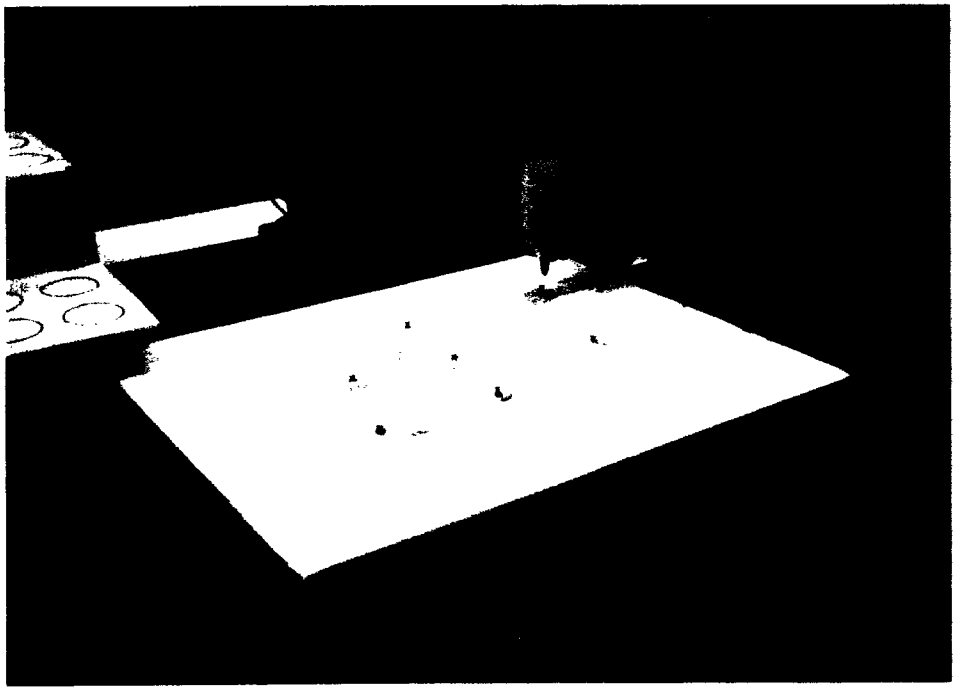
Model of shop floor to simulate the picking up of N number of objects.



Further research could produce a more advanced assembly facilities which combines several robots to automate virtually every aspect of production.



Sand is used to trace the optimal path taken by the robot.



To further optimize the path advanced research must be done in the area of obstacle collision and external sensing.