Florida Institute of Technology

Harris Institute for Assured Information

CSE 5280
Computer Graphics
Spring 2016

Class Assignment-02
(Animation – Robot Arm)

Student Name: **Jay Sandeepkumar Modi**

Student ID: **902292667**

Professor:

Dr. Eraldo Ribeiro

eribeiro@cs.fit.edu

Florida Institute of Technology

150 W. University Blvd, Melbourne, FL - 32901

http://www.fit.edu

1. Robot Arm:

Code:

```
function robotArm()
    close all;
    addpath( genpath( '.' ) );

    % Target position
    p = [ 10 15 ]';

    % Construction of basic body part and its local coordinate frame
    Part = BuildBasicPart();

    % Initial position of robot arm
    TransformedPart = DrawKinematicChain( Part, 0, 0 , 0, p );

    % Particle's initial position
    particle(1).StartPosition = TransformedPart.Joints(1:2,2);

    % Particle's current position
    particle(1).CurrentPosition = particle(1).StartPosition;

    % This is the goal position.
    particle(1).GoalPosition = p;

    % (x,y) coordinates of the centroid of obstacles
    o(1).location=[ 5 15 ]';
    o(1).R=3;
    o(2).location=[ 12 8 ]';
    o(2).R=3;
    o(3).location=[ 10 3 ]';
    o(3).R=3;

    % Advancement step for gradient descent
    lambda = 1;

    % Termination condition
    GoalReached = false;

    % Initialize variables for locations
    x = particle(1).StartPosition;
    g = particle(1).GoalPosition;

    % It repeats until goal is reached
    n=1;
    while ~GoalReached
        % Calculate next step using gradient descent
        x = x - lambda * Grad( x, g, o );

        hold on;
        temp = [x(1) x(2)];
```

```
        % Draw chain
        flag=0;
        for theta3 = 1 : 360
            for theta2 = 1 : 360
                for theta1 = 1 : 360
                    % taking decision whether the path is optimized or
                                                                    not
                    decision = estimation( Part, theta1, theta2 ,
                                                theta3, temp );
                    if decision == true
                        DrawKinematicChain( Part, theta1, theta2,
                                                    theta3, p );
                        flag = 1;
                    end

                end
                if flag == 1
                    break;
                end
            end
            if flag == 1
                break;
            end
        end

        % Getting each and every frame from figure and storing into GIF
        frame=getframe;
        im = frame2im(frame);
        [imind,cm] = rgb2ind(im,256);
        if n==1
            imwrite(imind,cm,'robot.gif','gif', 'Loopcount',inf);
            n=2;
        else
            imwrite(imind,cm,'robot.gif','gif','WriteMode','append');
        end

        % Pause for a moment so we can see the motion
        pause( .1 );

        % Check if goal has been reached.
        if ( norm( x - g ) <= 1 )
            GoalReached = true;
        end
    end

% Gradient vector (2-D direction) of cost function at current position
function G = Grad( p, g, o)

    y1 = p + [  0;  1 ];
    y2 = p + [  0; -1 ];

    x1 = p + [  1;  0 ];
    x2 = p + [ -1;  0 ];
```

```
    % Calculate the components of the gradient vector
    cx = Cpathplan( x1, g, o ) - Cpathplan( x2, g, o );
    cy = Cpathplan( y1, g, o ) - Cpathplan( y2, g, o );

    % Resultant vector formed by cost's x and y components
    r = [ cx; cy ];

    % Calculate the direction vector, i.e., direction of the gradient
      vector
    G = r / norm( r );

function c = Cpathplan( p, g, o )
    % Cost function for path planning calculated at position s with
      respect to
    % goal g and obstacle o

    % Value of log10E
    logE = 2.718281828;

    % Goal cost (Euclidean distance squared)
    c( 1 ) = norm( p - g );

    % Collision cost
    field=0;

    for i=1:size(o,2)
        dist = sqrt( (p(1)-o(i).location(1))^2 + ...
                     (p(2)- o(i).location(2))^2 );
        if 0 < dist && dist <= o(i).R
            field_temp = logE ^ (log( o(i).R / dist )* logE);
        else
            field_temp=0;
        end
        field = field + field_temp;
    end
    c( 2 ) = field;
    % Total cost
    c = c(1) + c(2);
return

function filledCylinder(x,y,r)
    [X,Y,Z] = cylinder(r);
    Z=-Z/12;
    surf(X+x,Y+y,Z)
return

function DrawPart( Part, color )
    % draw body part
    plot( Part.Pts( 1, : ), Part.Pts( 2, : ), [color '.-'],
      'LineWidth', 3 )
    axis( [ -5 25 -5 20 ] );
    view([15,66]);
    plot( Part.Joints( 1, : ), Part.Joints( 2, : ), ['r' '.'],
      'LineWidth', 3 )
return
```

```
function Part = BuildBasicPart()
    %-------------------------------------------------------------------
    %Construction of basic body part (all parts of same size and shape)
    %  -----------------------------------------------------------------

    % Basic body part of kinematic chain (rectangle)
    Part.Pts    = [ 0 0; 1 0; 1 1; 0 1; 0 0]';
    Part.Joints = [ 1/8 1/2; 7/8 1/2]';

    % Add a row of ones to Pts to convert to homogeneous coordinates
    Part.Pts    = [ Part.Pts; ...
                    ones( 1, size( Part.Pts, 2 ) ) ];
    Part.Joints = [ Part.Joints; ...
                    ones( 1, size( Part.Joints, 2 ) ) ];

    % Scale part horizontally to its final size.
    S = [ 8  0  0 ; ...
          0  2  0 ; ...
          0  0  1 ];

    Part.Pts    = S * Part.Pts;
    Part.Joints = S * Part.Joints;

    % Size of the scaled part (distance between joint points)
    Part.d = norm( Part.Joints( 1:2, 2 ) - Part.Joints( 1:2, 1 ) );

    % Place part's joint point at its rotation axis of the previous
      part.
    % This is done by translating the entire part so its "left" join
      point is
    % at the "right" joint point of the previous part. The base part
      will be
    % connected to the origin of the World coordinate system.
    t = -Part.Joints( 1:2, 1 ) ;

    T = [ 1  0  t(1)  ;...
          0  1  t(2)  ;...
          0  0   1   ];

    Part.Pts    = T * Part.Pts;
    Part.Joints = T * Part.Joints;

    % Draw the local coordinate system for the body part
    Part.x_axis = [ 0 0 1; 0 2 1]';
    Part.y_axis = [ 0 0 1; 2 0 1]';
return


% Rotate as a function of the angle
function R = Rotation( x )
    % degree-to-radian conversion
    theta = x * pi / 180;
```

```
    R = [ cos( theta )  -sin( theta )    0; ...
          sin( theta )   cos( theta )    0; ...
                0           0           1 ];
return


function T = Translation( d )
    % Translation by a vector [ d 0 ]
    T =  [ 1  0   d ; ...
           0  1   0 ; ...
           0  0   1 ];
return


function TransformedPart = DrawKinematicChain( Part, theta1, theta2,
                                                theta3, p )
    clf;
    hold on;

    %Drawing Obstacles (filled circles).
    filledCylinder(5,15,3);
    filledCylinder(12,8,3);
    filledCylinder(10,3,3);

    % Plot target point
    plot( p( 1 ), p( 2 ), 'ro', 'LineWidth', 2 );
    text( p( 1 )-1, p( 2 ) + 1, 'Target', 'FontSize', 13 );
    axis( [ -5 25 -5 20 ] );

    % Adding rotating rod initially.
    % Translate to join connection
    d =  0;
    R0 = Rotation(90);
    T0 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T0 * R0 * Part.Pts;
    TransformedPart.Joints  = T0 * R0 * Part.Joints;
    TransformedPart.x_axis  = T0 * R0 * Part.x_axis;
    TransformedPart.y_axis  = T0 * R0 * Part.y_axis;
    % draw body part
    DrawPart( TransformedPart, 'k' );

    % Translate to join connection
    d = Part.d;
    R1 = Rotation(theta1);
    T1 =  [ 1  0  0 ; ...
            0  1  d ; ...
            0  0  1 ];
    TransformedPart = Part;
    TransformedPart.Pts     = T1 * R1 * Part.Pts;
    TransformedPart.Joints  = T1 * R1 * Part.Joints;
    TransformedPart.x_axis  = T1 * R1 * Part.x_axis;
    TransformedPart.y_axis  = T1 * R1 * Part.y_axis;
    % draw body part
    DrawPart( TransformedPart, 'b' );
```

```
    % Translate to join connection
    d = Part.d;
    R2 = Rotation(theta2);
    T2 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T1 * R1 * T2 * R2 * Part.Pts;
    TransformedPart.Joints  = T1 * R1 * T2 * R2 * Part.Joints;
    TransformedPart.x_axis  = T1 * R1 * T2 * R2 * Part.x_axis;
    TransformedPart.y_axis  = T1 * R1 * T2 * R2 * Part.y_axis;
    % draw body part
    DrawPart( TransformedPart, 'b' );

    % Translate to join connection
    d = Part.d;
    R3 = Rotation(theta3);
    T3 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T1 * R1 * T2 * R2 * T3 * R3 * Part.Pts;
    TransformedPart.Joints  = T1 * R1 * T2 * R2 * T3 * R3 *
                                                    Part.Joints;
    TransformedPart.x_axis  = T1 * R1 * T2 * R2 * T3 * R3 *
                                                    Part.x_axis;
    TransformedPart.y_axis  = T1 * R1 * T2 * R2 * T3 * R3 *
                                                    Part.y_axis;
    % draw body part
    DrawPart( TransformedPart, 'b' );

    hold off;
return


function decision = estimation( Part, theta1, theta2, theta3, temp)
    % function for checking path is correct or not according to cost
    d =  0;
    R0 = Rotation(90);
    T0 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T0 * R0 * Part.Pts;
    TransformedPart.Joints  = T0 * R0 * Part.Joints;
    TransformedPart.x_axis  = T0 * R0 * Part.x_axis;
    TransformedPart.y_axis  = T0 * R0 * Part.y_axis;

    % Translate to join connection
    d = Part.d;
    R1 = Rotation(theta1);
    T1 =  [ 1  0   0 ; ...
            0  1   d ; ...
            0  0   1 ];

    %T1 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T1 * R1 * Part.Pts;
    TransformedPart.Joints  = T1 * R1 * Part.Joints;
    TransformedPart.x_axis  = T1 * R1 * Part.x_axis;
    TransformedPart.y_axis  = T1 * R1 * Part.y_axis;
```

```
    % Translate to join connection
    d = Part.d;
    R2 = Rotation(theta2);
    T2 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T1 * R1 * T2 * R2 * Part.Pts;
    TransformedPart.Joints  = T1 * R1 * T2 * R2 * Part.Joints;
    TransformedPart.x_axis  = T1 * R1 * T2 * R2 * Part.x_axis;
    TransformedPart.y_axis  = T1 * R1 * T2 * R2 * Part.y_axis;

    % Translate to join connection
    d = Part.d;
    R3 = Rotation(theta3);
    T3 = Translation(d);
    TransformedPart = Part;
    TransformedPart.Pts     = T1 * R1 * T2 * R2 * T3 * R3 * Part.Pts;
    TransformedPart.Joints  = T1 * R1 * T2 * R2 * T3 * R3 *
                                                    Part.Joints;
    TransformedPart.x_axis  = T1 * R1 * T2 * R2 * T3 * R3 *
                                                    Part.x_axis;
    TransformedPart.y_axis  = T1 * R1 * T2 * R2 * T3 * R3 *
                                                    Part.y_axis;

    check=(TransformedPart.Joints(1:2,2))';

    if round(check(1)) == round(temp(1)) &&
       round(check(2)) == round(temp(2))
        decision = true;
    else
        decision =false;
    end
return
```

Result: