Introduction to Robotics
Lab 9 - The Robot Games

Names: Hamad Abdul Razzaq & Muhammad Ali Siddiqui
IDs: hr06899 & ms06875

Lab Report

Task 9.1

Part (a)

The Jacobian loses its rank when:

$$a_2 a_3 \sin\theta_3 \left[a_2 \cos\theta_2 + a_3 \cos\left(\theta_2 + \theta_3\right)\right] = 0$$

This equation is satisfied when:

- $\theta_3 = 0$
- $\cos\left(\theta_2 + \theta_3\right) = -\cos\theta_2$

The following code verifies that the above two cases indeed makes the determinant of the Jacobian zero.

```
syms t
syms('theta_1(t)', 'theta_2(t)', 'theta_3(t)', 'theta_4(t)', 'alpha_1',
'alpha_2', ...
'alpha_3', 'alpha_4', 'a_1', 'a_2', 'a_3', 'a_4', 'd_1', 'd_2', 'd_3',
'd_4');
syms('T_01', 'T_12', 'T_23', 'T_34', 'T_04')
alpha_1 = pi/2; alpha_2 = 0; alpha_3 = 0; alpha_4 = 0;
d_1 = 5.4; d_2 = 0; d_3 = 0; d_4 = 0;
a_1 = 0; a_2 = 10.8; a_3 = 10.8; a_4 = 0;
T_01 = [cos(theta_1), -sin(theta_1)*cos(alpha_1), sin(theta_1)*sin(alpha_1),
a_1 * cos(theta_1);
sin(theta_1), cos(theta_1)*cos(alpha_1), -cos(theta_1)*sin(alpha_1), a_1 *
sin(theta_1);
0, sin(alpha_1), cos(alpha_1), d_1;
0 0 0 1];
TT_01 = T_01(t);
T_12 = [cos(theta_2), -sin(theta_2)*cos(alpha_2), sin(theta_2)*sin(alpha_2),
a_2 * cos(theta_2);
sin(theta_2), cos(theta_2)*cos(alpha_2), -cos(theta_2)*sin(alpha_2), a_2 *
sin(theta_2);
0, sin(alpha_2), cos(alpha_2), d_2;
0 0 0 1];
T_02 = T_01*T_12;
TT_02 = T_02(t);
T_23 = [cos(theta_3), -sin(theta_3)*cos(alpha_3), sin(theta_3)*sin(alpha_3),
a_3 * cos(theta_3);
sin(theta_3), cos(theta_3)*cos(alpha_3), -cos(theta_3)*sin(alpha_3), a_3 *
sin(theta_3);
0, sin(alpha_3), cos(alpha_3), d_3;
0 0 0 1];
T_03 = T_02*T_23;
TT_03 = T_03(t);
T_34 = [cos(theta_4), -sin(theta_4)*cos(alpha_4), sin(theta_4)*sin(alpha_4),
a_4 * cos(theta_4);
syms t
syms('theta_1(t)', 'theta_2(t)', 'theta_3(t)', 'theta_4(t)', 'alpha_1',
```

```
'alpha_2', ...
'alpha_3', 'alpha_4', 'a_1', 'a_2', 'a_3', 'a_4', 'd_1', 'd_2', 'd_3',
'd_4');
syms('T_01', 'T_12', 'T_23', 'T_34', 'T_04')
alpha_1 = pi/2; alpha_2 = 0; alpha_3 = 0; alpha_4 = 0;
d_1 = 5.4; d_2 = 0; d_3 = 0; d_4 = 0;
a_1 = 0; a_2 = 10.8; a_3 = 10.8; a_4 = 0;
T_01 = [cos(theta_1), -sin(theta_1)*cos(alpha_1), sin(theta_1)*sin(alpha_1),
a_1 * cos(theta_1);
sin(theta_1), cos(theta_1)*cos(alpha_1), -cos(theta_1)*sin(alpha_1), a_1 *
sin(theta_1);
0, sin(alpha_1), cos(alpha_1), d_1;
0 0 0 1];
TT_01 = T_01(t);
T_12 = [cos(theta_2), -sin(theta_2)*cos(alpha_2), sin(theta_2)*sin(alpha_2),
a_2 * cos(theta_2);
sin(theta_2), cos(theta_2)*cos(alpha_2), -cos(theta_2)*sin(alpha_2), a_2 *
sin(theta_2);
0, sin(alpha_2), cos(alpha_2), d_2;
0 0 0 1];
T_02 = T_01*T_12;
TT_02 = T_02(t);
T_23 = [cos(theta_3), -sin(theta_3)*cos(alpha_3), sin(theta_3)*sin(alpha_3),
a_3 * cos(theta_3);
sin(theta_3), cos(theta_3)*cos(alpha_3), -cos(theta_3)*sin(alpha_3), a_3 *
sin(theta_3);
0, sin(alpha_3), cos(alpha_3), d_3;
0 0 0 1];
T_03 = T_02*T_23;
TT_03 = T_03(t);
T_34 = [cos(theta_4), -sin(theta_4)*cos(alpha_4), sin(theta_4)*sin(alpha_4),
a_4 * cos(theta_4);
TOP3_X_3 = vpa(simplify(Jv(1:3,1:3)),3)
%% Solution#01
vv = subs(TOP3_X_3,theta_3,0)
d = vpa(simplify(det(vv)),3)
%% Solution#02
vv1 = simplify(subs(TOP3_X_3,cos(theta_2+theta_3),-cos(theta_2)));
vpa(simplify(det(vv1)),2)
```

Part (b)

- Position A: The arm will not be able to move along the axis along with the arm is fully strecthed.
- Position B: The arm will not be able to move along either the axis perpendicular to the plane formed by ARM with z axis.
- Position C: The arm will not be able to move along z-axis

Task 9.2

Code

```
function depth_example2()
%% Create all objects to be used in this file
% Make Pipeline object to manage streaming
pipe = realsense.pipeline();
% Make Colorizer object to prettify depth output
colorizer = realsense.colorizer();
% Create a config object to specify configuration of pipeline
cfg = realsense.config();


%% Set configuration and start streaming with configuration
% Stream options are in stream.m
streamType = realsense.stream('depth');
% Format options are in format.m
formatType = realsense.format('Distance');
% Enable default depth
cfg.enable_stream(streamType,formatType);
% Enable color stream
streamType = realsense.stream('color');
formatType = realsense.format('rgb8');
cfg.enable_stream(streamType,formatType);

% Start streaming on an arbitrary camera with chosen settings
profile = pipe.start();

%% Acquire and Set device parameters
% Get streaming device's name
dev = profile.get_device();
name = dev.get_info(realsense.camera_info.name);

% Access Depth Sensor
depth_sensor = dev.first('depth_sensor');

% Access RGB Sensor
rgb_sensor = dev.first('roi_sensor');

% Find the mapping from 1 depth unit to meters, i.e. 1 depth unit =
% depth_scaling meters.
depth_scaling = depth_sensor.get_depth_scale();

% Set the control parameters for the depth sensor
% See the option.m file for different settable options that are visible
% to you in the viewer.
optionType = realsense.option('visual_preset');
% Set parameters to the midrange preset. See for options:
% ...
   https://intelrealsense.github.io/librealsense/doxygen/rs__option_8h.html#a07402b9eb861d1
depth_sensor.set_option(optionType,9);

% Set autoexposure for RGB sensor
optionType = realsense.option('enable_auto_exposure');
rgb_sensor.set_option(optionType,1);
optionType = realsense.option('enable_auto_white_balance');
rgb_sensor.set_option(optionType,1);

%% Align the color frame to the depth frame and then get the frames
```

```
    % Get frames. We discard the first couple to allow
    % the camera time to settle
    for i = 1:5
        fs = pipe.wait_for_frames();
    end

    % Alignment
    align_to_depth = realsense.align(realsense.stream.depth);
    fs = align_to_depth.process(fs);

    % Stop streaming
    pipe.stop();

    %% Depth Post-processing
    % Select depth frame
    depth = fs.get_depth_frame();
    width = depth.get_width();
    height = depth.get_height();

    % Decimation filter of magnitude 2
%     dec = realsense.decimation_filter(2);
%     depth = dec.process(depth);

    % Spatial Filtering
    % spatial_filter(smooth_alpha, smooth_delta, magnitude, hole_fill)
    spatial = realsense.spatial_filter(.5,20,2,0);
    depth_p = spatial.process(depth);

    % Temporal Filtering
    % temporal_filter(smooth_alpha, smooth_delta, persistence_control)
    temporal = realsense.temporal_filter(.13,20,3);
    depth_p = temporal.process(depth_p);

    %% Color Post-processing
    % Select color frame
    color = fs.get_color_frame();

    %% Colorize and display depth frame
    % Colorize depth frame
    depth_color = colorizer.colorize(depth_p);

    % Get actual data and convert into a format imshow can use
    % (Color data arrives as [R, G, B, R, G, B, ...] vector)fs
    data = depth_color.get_data();
    img = ...
        permute(reshape(data',[3,depth_color.get_width(),depth_color.get_height()]),[3 ...
        2 1]);

    % Display image
%     save img
    imshow(img);
    title(sprintf("Colorized depth frame from %s", name));

    %% Display RGB frame
    % Get actual data and convert into a format imshow can use
    % (Color data arrives as [R, G, B, R, G, B, ...] vector)fs
    data2 = color.get_data();
```

```
    im = permute(reshape(data2',[3,color.get_width(),color.get_height()]),[3 2 1]);
    color.get_width()
    color.get_height()
    size(im)

    % Display image
    figure;
%     save im
    imshow(im);
    title(sprintf("Color RGB frame from %s", name));

    %% Depth frame without colorizing
    % Convert depth values to meters

    data3 = depth_scaling * single(depth_p.get_data());

    %Arrange data in the right image format
    ig = permute(reshape(data3',[width,height]),[2 1]);

    % Scale depth values to [0 1] for display
%     save ig
    figure;
    imshow(mat2gray(ig));

%% Processing RGB Image - Lab 7
BW = imbinarize(im); % Binarizing the given rgb image
figure; % Displaying Binarized Image along with original rgb image
imshowpair(im,BW,'montage');
title('RGB Image & its binarized version');
len = size(BW); % Size of image
BW2 = ~(BW(:,:,1) & BW(:,:,2) & BW(:,:,3)); % This expression is used to assign the ...
   color black to the background that white part in the rgb image. And it assigns ...
   white to the non-white part in the rgb image. This is done to differentiate the ...
   background with the objects in the scene.
BW3 = BW2;
figure; % Displaying the obtained black and white image that differentiates background ...
   from objects
imshow(BW3);
title('Inverted Binarized Image');
cc8 = bwconncomp(BW3); % This function finds all the connected components, i.e., ...
   objects in the scene and stores the pixel indices for each object
remove_idx = []; % This array will contain the pixel indices of those objects which ...
   are not cube
% Filtering out stuff other than cubes
for i = 1:length(cc8.PixelIdxList)
    if (length(cc8.PixelIdxList{i}) < 200) || (length(cc8.PixelIdxList{i}) > 2000) % ...
       For any object containing less than 200 pixels or more than 2000 pixels, they ...
       cannot be cubes. So indices of those pixels are added to remove_idx
        remove_idx(end + 1) = i;
    end
end
for i=1:length(remove_idx)
    BW2(cc8.PixelIdxList{remove_idx(i)}) = 0; % Setting the value of the pixel indices ...
       corresponding to non-cube objects to 0, i.e. turning them black which is ...
       equivalent to removing them.
end
figure;
```

```
imshowpair(BW3, BW2, 'montage'); % Displaying Black & White image after removing the ...
    non-cube objects
title('Before & After');
cc8 = bwconncomp(BW2); % Finding the connected components in the new image so that we ...
    only have the information of cubes stored
BW4 = BW2;
object_info = struct; % Making a structure that contains all the information from our ...
    image processing
object_info.Index_data = cc8.PixelIdxList; % Object Indices
object_info.rgb_image = im2double(im); % RGB Image
object_info.depth_info_cm = ig*100; % Depth Info in cm
object_info.red_val = zeros(1,length(object_info.Index_data)); % Avg Red for each obj
object_info.green_val = zeros(1,length(object_info.Index_data)); % Avg Green for each obj
object_info.blue_val = zeros(1,length(object_info.Index_data)); % Avg Blue for each obj
object_info.red_val_max = zeros(1,length(object_info.Index_data)); % Max Red for each obj
object_info.green_val_max = zeros(1,length(object_info.Index_data)); % Max Green for ...
    each obj
object_info.blue_val_max = zeros(1,length(object_info.Index_data)); % Max Blue for ...
    each obj
object_info.top_data = object_info.Index_data; % Indices for top faces of each object
object_info.top_data_width = object_info.Index_data; % x Indices for top faces of each ...
    object
object_info.top_data_column = object_info.Index_data; % y Indices for top faces of ...
    each object
object_info.no_top_data = object_info.Index_data; % This will store all the indices of ...
    the object that is not a top face pixel
object_info.center_x = zeros(1, length(object_info.Index_data)); % x value of the ...
    center pixel of each object
object_info.center_y = zeros(1, length(object_info.Index_data)); % y value of the ...
    center pixel of each object
object_info.red_val_bin = zeros(1,length(object_info.Index_data)); % A Binary array ...
    telling whether each object contains red color or not
object_info.green_val_bin = zeros(1,length(object_info.Index_data)); % A Binary array ...
    telling whether each object contains green color or not
object_info.blue_val_bin = zeros(1,length(object_info.Index_data)); % A Binary array ...
    telling whether each object contains blue color or not
object_info.depth_val = zeros(1,length(object_info.Index_data));
for i = 1:length(object_info.Index_data) % Looping over each object
    arr = []; % This array contains those pixels of an object whose depth value is ...
        non-zero
    for j = 1:length(object_info.Index_data{i})
        if object_info.depth_info_cm(object_info.Index_data{i}(j)) ~= 0
            arr(end +1) = object_info.depth_info_cm(object_info.Index_data{i}(j));
        end
    end
    counts = hist(arr, 10); % A Histogram is generated on the pixels having non-zero ...
        depth values
    th = min(arr) + (otsuthresh(counts)*(max(arr) - min(arr))); % Determining ...
        threshold for top face detection using otsu algorithm
    object_info.depth_val(i) = th;
    r = 0;
    l = 0;
    for j = 1:length(object_info.Index_data{i})

        if object_info.depth_info_cm(object_info.Index_data{i}(j)) > th % Separating ...
            top face indices using this condition
            BW4(object_info.Index_data{i}(j)) = 0;
```

```
            object_info.top_data{i}(j-r) = [];
            r = r + 1;
        else
            object_info.no_top_data{i}(j-l) = [];
            l = l + 1;
        end
    end
end

for i = 1:length(object_info.top_data)
    object_info.top_data_column{i} = floor(object_info.top_data{i} ./ len(1)) + 1;
    object_info.top_data_width{i} = mod(object_info.top_data{i}, len(1));
    object_info.center_x(i) = floor(mean(object_info.top_data_width{i})); % ...
       Calculating x value of Center pixel
    object_info.center_y(i) = floor(mean(object_info.top_data_column{i}));% ...
       Calculating y value of Center pixel
    BW4(object_info.center_x(i), object_info.center_y(i)) = 0;  % Setting color for ...
       center value to be black so that it is visible
end
figure;
imshowpair(BW2, BW4, 'montage'); % Showing the objects with both top face and non-top ...
   face pixels and objects with only top face pixels
title('Face top detection');
avg_th = 0.3; % Threshold for average RGB value
max_th = 0.4; % Threshold for maximum RGB value
for k = 1:length(object_info.top_data_width) % Looping for every object
    % Finding the average and maximum values
    val_r = 0;
    val_g = 0;
    val_b = 0;
    max_r = -inf;
    max_g = -inf;
    max_b = -inf;
    for i = 1:length(object_info.top_data_width{k})
            if ...
               (object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_col
               > max_r)
                max_r = ...
                   object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_
            end
            if ...
               (object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_col
               > max_g)
                max_g = ...
                   object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_
            end
            if ...
               (object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_col
               > max_b)
                max_b = ...
                   object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_
            end
            val_r = val_r + ...
               object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_colu
            val_g = val_g + ...
               object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_colu
```

```
            val_b = val_b + ...
                object_info.rgb_image(object_info.top_data_width{k}(i),object_info.top_data_colu
    end
    object_info.red_val(k) = val_r / length(object_info.top_data_width{k});
    object_info.green_val(k) = val_g / length(object_info.top_data_width{k});
    object_info.blue_val(k)= val_b / length(object_info.top_data_width{k});
    object_info.red_val_max(k) = max_r;
    object_info.green_val_max(k) = max_g;
    object_info.blue_val_max(k) = max_b;
    % Setting RGB values if the required thresholds are met
    if object_info.red_val(k) >= avg_th && object_info.red_val_max(k) > max_th
        object_info.red_val_bin(k) = 1;
    end
    if object_info.green_val(k) >= avg_th && object_info.green_val_max(k) > max_th
        object_info.green_val_bin(k) = 1;
    end
    if object_info.blue_val(k) >= avg_th && object_info.blue_val_max(k) > max_th
        object_info.blue_val_bin(k) = 1;
    end
end
object_info
object_info.depth_val
imshow(im);
save lab9b3
%% Conversion of Red and Yellow Object Coordinates into World Coordinates - Lab 8 Modified
% Extracting Red and Yellow Objects
red_uv = [];
yellow_uv = [];
for i = 1:length(object_info.red_val_bin)
    if (object_info.red_val_bin(i) && ~(object_info.green_val_bin(i)) && ...
       ~(object_info.blue_val_bin(i)))
        red_uv(end + 1) = [object_info.center_x(i); object_info.center_y(i); 1];
    elseif (object_info.red_val_bin(i) && (object_info.green_val_bin(i)) && ...
       ~(object_info.blue_val_bin(i)))
        yellow_uv(end + 1) = [object_info.center_x(i); object_info.center_y(i); 1];
    end
end
% Converting Red and Yellow Pixel values to world coordinates
WF_coord_red = [];
WF_coord_yellow = [];
Z = 820;
K = [
    1399.1 1 944.4568 ;
    0 1399.1 533.8895;
    0 0 1
    ];
T_ = [
    1 0 0 -20;
    0 cos(pi) -sin(pi) 0;
    0 sin(pi) cos(pi) 688;
    ];
for i = (length(red_uv))
    WF_coord_red(end + 1) = inv(K * T_) * red_uv(i)*Z;
end
for i = (length(yellow_uv))
    WF_coord_yellow(end + 1) = inv(K * T_) * yellow_uv(i)*Z;
end
```

```
% Sending the coordinates  one by one to FSM of Lab 6 (with modification)
global obj_no_red;
global obj_no_yellow;
obj_no_red = 1;
obj_no_yellow = 1;
for i = (length(red_uv))
    FSM(WF_coord_red(i), true);
end
for i = (length(yellow_uv))
    FSM(WF_coord_yellow(i), true);
end
end
```

In the above code, the functin FSM is as follows:

```
function out = FSM(Pick, red)
    State = 0;
    global arb;
    global obj_no_red;
    global obj_no_yellow;
    arb = Arbotix('port', 'COM4', 'nservos', 5);
    temp = arb.getpos();
    arb.setpos(temp, [64 64 64 64 64]);
    motor_speeds = [64 64 64 64 64];
    pause(2);
    Place_Pos_red = [18 18; 18 13; 18 8; 18 3;];
    Place_Pos_yellow = [ 18 -2; 18 -7; 18 -12; 18 -17];
    global grip_val;
    grip_val = 0;
    while 1
        if (State == 0)
            setPosition([pi/4 pi/4 pi/4 pi/4], grip_val);
            pause(3);
            x = 0; y = -21.6; z = -2; phi = -pi/2;
            disp('Pick Position:');
            disp(Pick);
            State = 1;
        elseif (State == 1)
            disp('State:');
            disp(State);
            Pick(3) = Pick(3) + 3;
            theta_arr = findOptimalSolution(Pick(1), Pick(2), Pick(3), Pick(4));
            setPosition(theta_arr, grip_val);
            pause(2);
            State = 2;
        elseif State == 2
            disp('State:');
            disp(State);
            Pick(3) = Pick(3) - 3;
            theta_arr = findOptimalSolution(Pick(1), Pick(2), Pick(3), Pick(4));
            setPosition(theta_arr, grip_val);
            pause(2);
            State = 3;

        elseif State == 3
            grip_val = 1.2;
```

```
        disp('State:');
        disp(State);
        b = arb.getpos();
        b(5) = grip_val;
        arb.setpos(b, motor_speeds);
        pause(3);
        State = 4;
    elseif State == 4
        disp('State:');
        disp(State);
        Pick(3) = Pick(3) + 6;
        theta_arr = findOptimalSolution(Pick(1), Pick(2), Pick(3), Pick(4));
        setPosition(theta_arr, grip_val);
        pause(2);
        State = 5;
    elseif State == 5
        disp('State:');
        disp(State);

        if (red)
            Place = Place_Pos_red(obj_no_red);
            obj_no_red = obj_no_red + 1
        else
            Place = Place_Pos_yellow(obj_no_yellow);
            obj_no_yellow = obj_no_yellow + 1;
        end
        State = 6;
    elseif State == 6
        disp('State:');
        disp(State);
        Place(3) = Place(3) + 3;
        theta_arr = findOptimalSolution(Place(1), Place(2), Place(3), Place(4));
        setPosition(theta_arr, grip_val);
        pause(2);
        State = 7;
    elseif State == 7
        disp('State:');
        disp(State);
        Place(3) = Place(3) - 5;
        theta_arr = findOptimalSolution(Place(1), Place(2), Place(3), Place(4));
        setPosition(theta_arr, 1.2);
        findPincher(theta_arr);
        pause(2);
        State = 8;
    elseif State == 8
        grip_val = 0;
        disp('State:');
        disp(State);
        b = arb.getpos();
        b(5) = grip_val;
        arb.setpos(b, [64 64 64 64 64]);
        pause(2);
        State = 9;
    elseif State == 9
        disp('State:');
        disp(State);
        Place(3) = Place(3) + 5;
```

```
            theta_arr = findOptimalSolution(Place(1), Place(2), Place(3), Place(4));
            setPosition(theta_arr, grip_val);
            pause(1);
            State =10;
        elseif State == 10
            disp('State:');
            disp(State);
            arb.setpos([pi/4 pi/4 pi/4 pi/4 grip_val], [64 64 64 64 64]);
            break
        end

    end
end
```

Task 9.3

Code

The changes in the relevant part of the above code is added here:

```
    %% Conversion of Red and Yellow Object Coordinates into World Coordinates - Lab 8 ...
       Modified
% Extracting Red and Yellow Objects
red_uv = [];
red_depth = [];
yellow_uv = [];
yellow_depth = [];
for i = 1:length(object_info.red_val_bin)
    if (object_info.red_val_bin(i) && ~(object_info.green_val_bin(i)) && ...
       ~(object_info.blue_val_bin(i)))
        red_uv(end + 1) = [object_info.center_x(i); object_info.center_y(i); 1];
        red_depth(end + 1) = object_info.depth_val(i);
    elseif (object_info.red_val_bin(i) && (object_info.green_val_bin(i)) && ...
       ~(object_info.blue_val_bin(i)))
        yellow_uv(end + 1) = [object_info.center_x(i); object_info.center_y(i); 1];
        yellow_depth(end + 1) = object_info.depth_val(i);
    end
end
% Converting Red and Yellow Pixel values to world coordinates
WF_coord_red = [];
WF_coord_yellow = [];
Z = 820;
K = [
    1399.1 1 944.4568 ;
    0 1399.1 533.8895;
    0 0 1
    ];
T_ = [
    1 0 0 -20;
    0 cos(pi) -sin(pi) 0;
    0 sin(pi) cos(pi) 688;
    ];
for i = (length(red_uv))
    WF_coord_red(end + 1) = inv(K * T_) * red_uv(i)*Z;
```

```
end
for i = (length(yellow_uv))
    WF_coord_yellow(end + 1) = inv(K * T_) * yellow_uv(i)*Z;
end
% Sending the coordinates  one by one to FSM of Lab 6 (with modification)
global obj_no_red;
global obj_no_yellow;
obj_no_red = 1;
obj_no_yellow = 1;
for i = (length(red_uv))
    WF_coord_red(3, i) = red_depth(i);
    FSM(WF_coord_red(i), true);
end
for i = (length(yellow_uv))
    WF_coord_yellow(3, i) = yellow_depth(i);
    FSM(WF_coord_yellow(i), true);
end
```

Analysis of Task 9.2 & Task 9.3

The logic for the above code is working perfectly fine. However, we were not able to make a proper demo again because of some minimal errors. As a result this task cannot be achieved due to following reasons.

- **Robot reaching close to the Cube but unable to pick it**: There seems to be a very small error in either determining the center pixel of the cube or applying inverse kinematics to find the joint angles. This is because our manipulator was reaching nearer to the cube (indicating the correctness of our approach) but was unable to reach exactly the center of the pixel (which indicates the error involved).
- **Change of WorkSpace**: The lighting in our workspace stopped working and as a result we had to move to a different workspace and re do the work for lab 7 based on the new lighting conditions. This ate up more time
- **Time Constraint**: As there were final exams going on with ongoing projects and the above mentioned technicalities, we were not able to give more time to the lab and hance couldn't figure out how to eliminate error. But atleast, it was verified that the logic is working fine since the robot was moving nearer to the desired location every time. Accuracy was an issue which we were not able to fix in the given time