BRADLEY
UNIVERSITY

ECE 467: Robotics Design Laboratory

Lab 5: Leader-Follower Formation Control With Pioneer 3-DX Robots

Anthony Le, Mike McGrath, and Jason Morris

Contact Information

Anthony Le
Electrical and Computer Engineering Department
Caterpillar College of Engineering and Technology
Bradley University
Jobst Hall #254
1501 W. Bradley Avenue
Peoria, IL, 61625, USA
Phone: +1 (309) 369-8462
e-Mail: ale@mail.bradley.edu

Table of Contents

1 Objective(s)

In this lab, our goal was to learn formation control algorithms using nonlinear feedback control. We wanted to also learn how to use the robot's operating system to be able to control the robot wirelessly. We then must implement a leader-follower algorithm using two separate Pioneer 3-DX robots.

2 Background

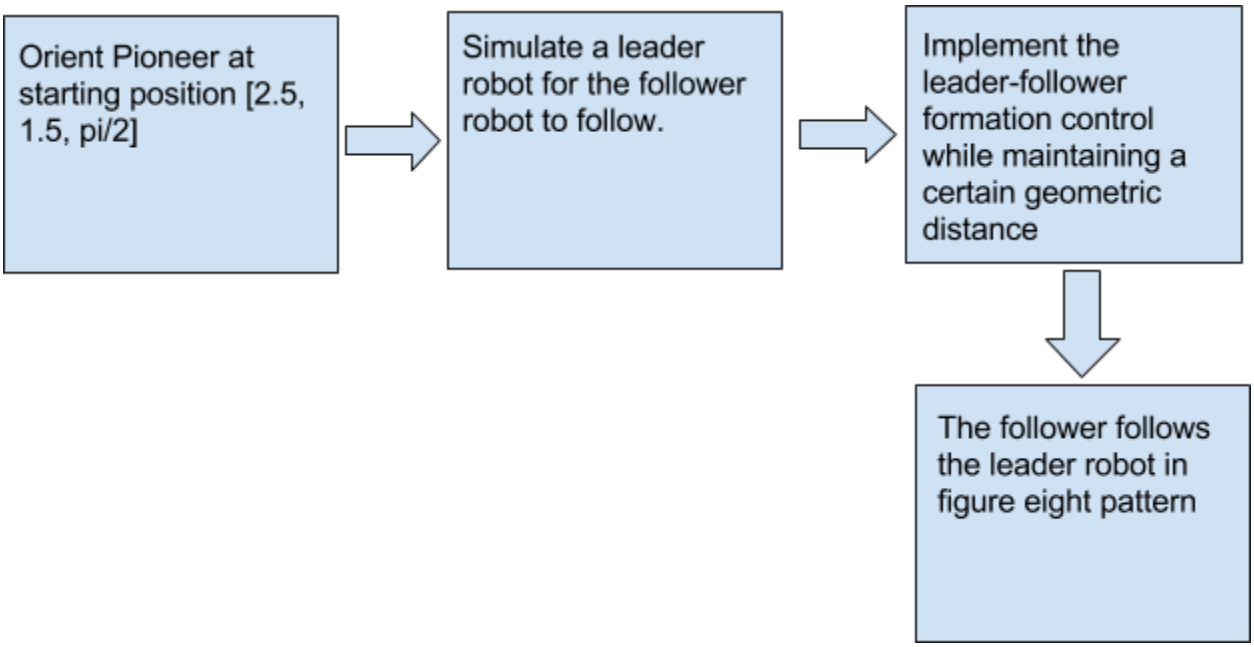
2.1 Software

- Adept MobileRobots Advanced Robotics Interface for Applications (ARIA)
- Matlab 2015
- Robotics System Toolbox for Matlab
- V-REP
- Robotics Operating System (ROS)

2.2 Hardware

- Pioneer P3-DX robot (2)
- Laptop
- USB to Serial adapter
- BeagleBone Black
- USB WiFi adapter

2.3 Block Diagram



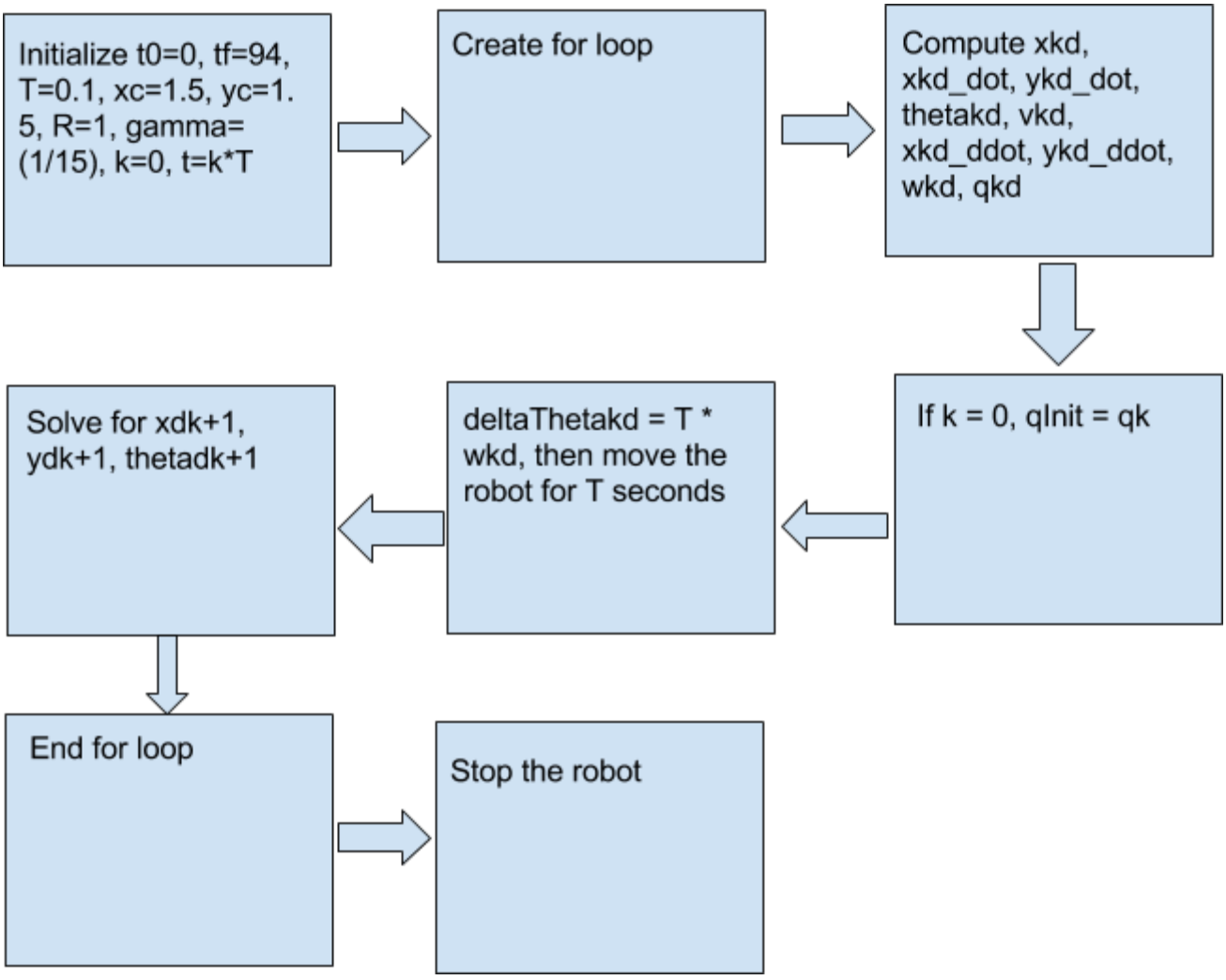
3 Implementation

3.1 Steps

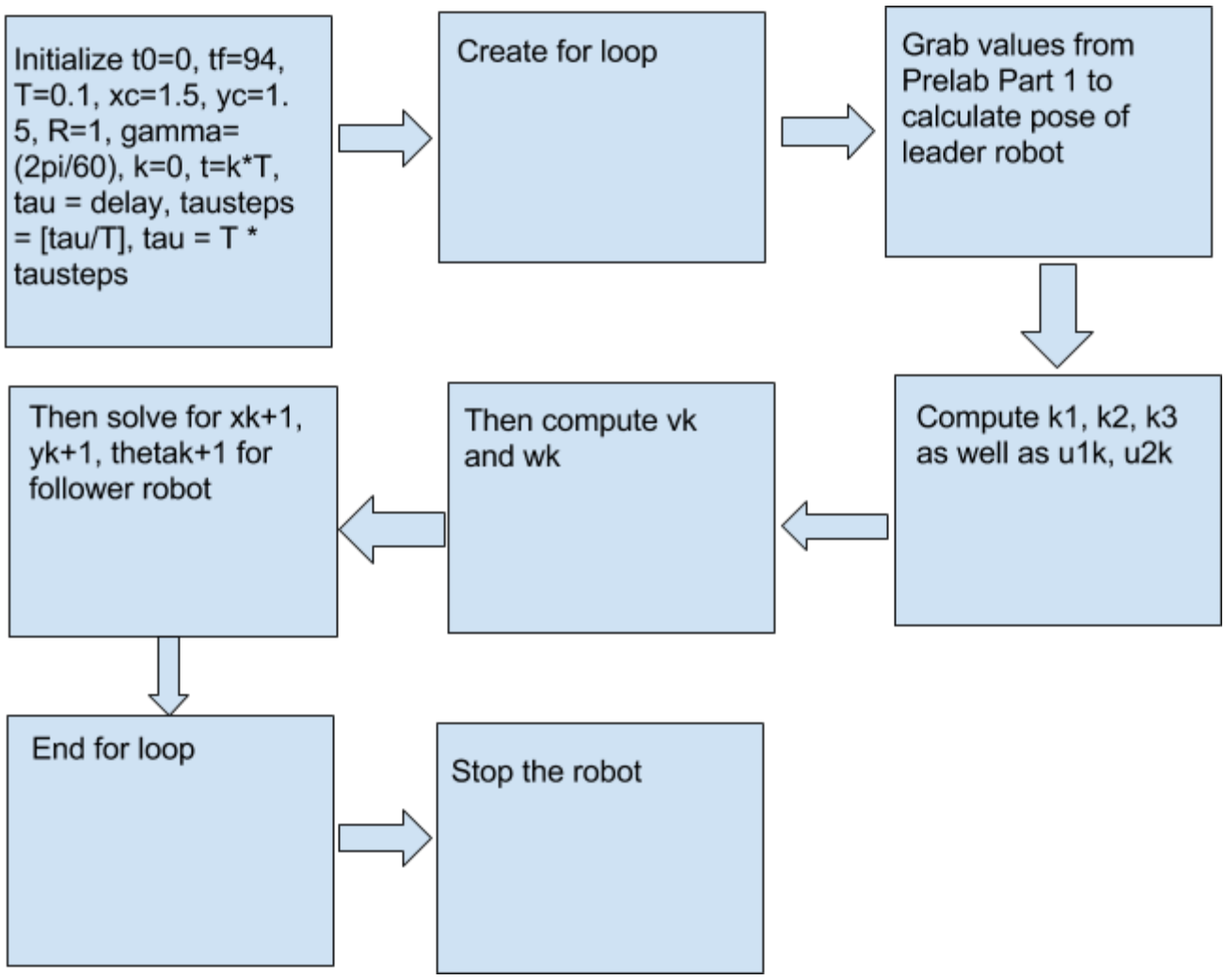
1. Used implementation steps given in class to create prelab code for part 1.
2. With example code given on Sakai, we were able to implement prelab code to wirelessly control the robot.
3. Simulated part 1 code on V-REP
4. Tweaked part 1 prelab code along with implementation steps given in class to create prelab code for part 2.
5. Combine prelab 2 code with example code from Sakai to wirelessly control robot.
6. Simulated part 2 code on V-REP
7. Demonstrated final code.

3.2 Code

Part 1 Flow Chart



Part 2 Flow Chart



MATLAB CODE

- PRELAB 1

```
function [desired, vdt, omegadt] = Prelab5Part1
% Created by S. Miah on Feb. 01, 2016.
close all
clear all
clc

disp('Please wait while the simulation runs ...');

% ---------------------
% SIMULATION PARAMETERS
% ---------------------

% Simulation time
t0 = 0; tf = 94; % initial and final simulation time [s]
T = 0.1;  % Sampling time [s]
tsteps = floor((tf-t0)/T); % number of time steps
dt = T*(0:tsteps)'; % Discrete time vector (include time t0 to plot
initial state too)
gamma = (1/15);


% Generate reference trajectory (circle)
R1 = 1; % radius of the circle [m]
R2 = 1;
center = [1.5 1.5]; % [m]

for k = 0:1:tsteps

    t = k*T;
    xdt = center(1) + R1*sin(2*gamma*t);
    ydt = center(2) + R2*sin(gamma*t);

    xd_dott = 2*R1*gamma*cos(2*gamma*t);
    yd_dott = (gamma)*(R2)*cos(gamma*t);
    thetadt = atan2(yd_dott, xd_dott);

    xd_dot_dott = -4*(gamma.^2)*R1*sin(2*gamma*t);
    yd_dot_dott = ((gamma.^2)*(-R2)*sin(gamma*t));

    vdt(k+1, 1) = sqrt(xd_dott.^2 + yd_dott.^2);
    tmp = yd_dot_dott.*xd_dott - xd_dot_dott.*yd_dott;
    omegadt(k+1, 1) = tmp./(vdt(k+1, 1).^2);
```

```
    qk = [xdt,ydt,thetadt];

        if k == 0
        desiredPose(k+1, :) = qk;
        end

    delta_theta = T*omegadt(k+1,1);
    xdtnew = xdt + T*vdt(k+1, 1)*cos(thetadt + (delta_theta/2));
    ydtnew = ydt + T*vdt(k+1, 1)*sin(thetadt + (delta_theta/2));
    thetadtnew = thetadt + delta_theta;

    if(thetadtnew>pi)
        thetadtnew = thetadtnew - (2*pi);
    end

    if(thetadtnew<-pi)
        thetadtnew = thetadtnew + (2*pi);
    end

    qknew = [xdtnew,ydtnew,thetadtnew];

    desiredPose(k+2, :) = qknew;

end

desired = desiredPose;

% jack = vdt;
% generatePlots(length(dt), dt, [], desiredPose, [], [vdt omegadt]);


function generatePlots(Tn, t, actualStates, desiredStates, error, u)
    close all; % close all opened figures
    % Tn = number of discrete time/path parameter points
    % t = time/path parameter history, dimension = Tn x 1
    % actualState = actual state of the system, dimension = Tn x n
    % desiredState = desired state of the system, dimension = Tn x n
    % error = desired state - actual state, dimension =  Tn x n
    % u = control inputs, dimension = Tn x m


    % Plot the robot's  velocities,
    figure
    subplot(2,1,1)
    plot(t,u(:,1), 'k-','LineWidth', 1.5);
```

```
    xlabel('Time [s]');
    ylabel('Linear speed [m/s]');
    grid on

    subplot(2,1,2)
    plot(t,u(:,2), 'k--','LineWidth', 1.5);
    xlabel('Time [s]');
    ylabel('Angular speed [rad/s]');
    grid on

    savefilename = ['OUT/controlInputs'];
    saveas(gcf, savefilename, 'fig');
    print('-depsc2', '-r300', [savefilename, '.eps']);

    % Create a movie of the simulation (path/trajectory)


    xmax = max(desiredStates(:,1));
    xmin = min(desiredStates(:,1));
    ymax = max(desiredStates(:,2));
    ymin = min(desiredStates(:,2));

    vid = VideoWriter('OUT/trajectory.avi');
    vid.Quality = 100;
    vid.FrameRate = 5;
    open(vid)

    fig=figure;
    clf reset;

    for i = 1:Tn,
        clf;
        box on;
        axis([xmin-5 xmax+5 ymin-5 ymax+5]);
        axis equal;
        axis manual;
        [Xa,Ya] = plot_DDMR(desiredStates(i,:),axis(gca)); %
        hold on;
        desired =
plot(desiredStates(1:i,1),desiredStates(1:i,2),'LineWidth',1.5);
        hold on
        %actual = plot(actualStates(1:i,1),actualStates(1:i,2),'k--');
        fill(Xa,Ya,'r');
        hold off;
        xlabel('x [m]');
```

```
        ylabel('y [m]');
        F = getframe(fig);
        writeVideo(vid,F);
    end
    [Xa,Ya] = plot_DDMR(desiredStates(1,:),axis(gca)); % DDMR =>
Differential drive mobile robot
    grid on
    hold on
    plot(Xa,Ya);
    hold on
    %legend([actual desired],'actual', 'desired');
    savefilename = 'OUT/trajectory';
    saveas(gcf, savefilename, 'fig');
    print('-depsc2', '-r300', [savefilename, '.eps']);

    close(vid);

    % Create the movie and save it as an AVI file
    % winopen('OUT/trajectory.avi')



    function [X,Y] = plot_DDMR(Q,AX)
% PLOT_UNICYCLE   Function that generates lines for plotting a unicycle.
%
%    PLOT_UNICYCLE(Q,AX) takes in the axis AX = axis(gca) and the unicycle
%    configuration Q and outputs lines (X,Y) for use, for example, as:
%       fill(X,Y,'b')
%    This must be done in the parent script file.

x     = Q(1);
y     = Q(2);
theta = Q(3);

l1 = 0.02*max([AX(2)-AX(1),AX(4)-AX(3)]);
X = [x,x+l1*cos(theta-2*pi/3),x+l1*cos(theta),x+l1*cos(theta+2*pi/3),x];
Y = [y,y+l1*sin(theta-2*pi/3),y+l1*sin(theta),y+l1*sin(theta+2*pi/3),y];
```

- PRELAB 2

```
function Prelab5Part2code
% Created by J. Morris on Apr. 12, 2016.
close all
clear all
clc
```

```
disp('Please wait while the simulation runs ...');

% ---------------------
% SIMULATION PARAMETERS
% ---------------------

% Simulation time
t0 = 0; tf = 94; % initial and final simulation time [s]
T = 0.1;  % Sampling time [s]
tsteps = floor((tf-t0)/T); % number of time steps
dt = T*(0:tsteps)'; % Discrete time vector (include time t0 to plot
initial state too)
gamma = (1/15);
zeta = .7;
b = 1;
a = 1;

Tau = 5;
Tausteps = floor(Tau/T);
Tau = T*(0:Tausteps)';

qInit = [1, 1, (pi/2)];
% qInit = [1.5, 1.5, .464];
qkprnew = qInit;

[desiredPose, vddt, omegaddt] = Prelab5Part1();

    numit = length(desiredPose(:, 1));

    vddt(numit, :) = vddt(numit-1, :);
    omegaddt(numit, :) = omegaddt(numit-1, :);

for i = 1:(Tausteps)
desiredPose(numit+i,:) = desiredPose(numit,:);
vddt(numit+i,:) = vddt(numit,:);
omegaddt(numit+i,:) = omegaddt(numit,:);
end

xdes = desiredPose(:, 1);
ydes = desiredPose(:, 2);
thetades = desiredPose(:, 3);

% k1 = (2*zeta*a);
k1 = (6);
k2 = b;
```

```
k3 = k1;

% disp((tsteps+Tausteps+1))
% disp(length(vddt))

for k = 0:tsteps+Tausteps+1

    t = k*T;
    dtnew(k+1,1) = t;
    actualPose(k+1, :) = qkprnew;

    if(k >= Tausteps)
        kpr = k - Tausteps + 1;

        e_xpr = ((xdes(kpr, 1)-(actualPose(k+1,1)))*cos(actualPose(k+1,
3)))+((ydes(kpr, 1)-(actualPose(k+1,2)))*sin(actualPose(k+1, 3)));
        e_ypr = (((-(xdes(kpr, 1)))-actualPose(k+1,1))*sin(actualPose(k+1,
3)))+((ydes(kpr, 1)-actualPose(k+1,2))*cos(actualPose(k+1, 3)));
        e_thetapr = thetades(kpr, 1) - actualPose(k+1,3);

        u1pr = (-k1*e_xpr);
        u2pr = (-k2*vddt(k+1,
1)*(((sin(e_thetapr)))/(e_thetapr))*e_ypr)-(k3*e_thetapr);

        vkpr(k+1, 1) = (vddt(kpr, 1)*cos(e_thetapr))-u1pr;
        omegadtpr(k+1, 1) = omegaddt(kpr, 1) - u2pr;

        delta_thetapr = (T*omegadtpr(k+1,1));

        xkprnew = actualPose(k+1, 1)+(T*vkpr(k+1,1)*cos(actualPose(k+1,
3)+(delta_thetapr/2)));
        ykprnew = actualPose(k+1, 2)+(T*vkpr(k+1,1)*sin(actualPose(k+1,
3)+(delta_thetapr/2)));

        thetakprnew = actualPose(k+1, 3) + delta_thetapr;

        if(thetakprnew>pi)
        thetakprnew = thetakprnew - (2*pi);
        end

        if(thetakprnew<-pi)
        thetakprnew = thetakprnew + (2*pi);
        end

        qkprnew = [xkprnew,ykprnew,thetakprnew];
```

```
%         actualPose(k+2, :) = qkprnew;

    end
end


% for i = 1:3
% newDesired(:,i) = desiredPose(:,i);
% end

poseError = desiredPose-actualPose;


generatePlots(length(dtnew), dtnew, actualPose, desiredPose, poseError,
[vkpr omegadtpr]);
% generatePlots(length(dtnew), dtnew, [], actualPose, [], [vddt
omegaddt]);


disp('... done.');


function edot = stateEqError(t,e,k1,k2,k3,dt,vdt,omegadt)

omegad = interp1(dt, omegadt, t);
k2t = interp1(dt, k2, t);
vd = interp1(dt, vdt, t);

At=[-k1 omegad 0;
    -omegad 0 vd;
    0 -k2t -k3];

edot = At*e;


% Reference robot kinematic model
function qddot = stateEqDesired(t,qd,dt,vdt,omegadt)

vd = interp1(dt,vdt,t);
omegad = interp1(dt,omegadt,t);
thetad = qd(3);
xddott = vd*cos(thetad);
yddott = vd*sin(thetad);
thetaddott = omegad;
```

```
qddot = [xddott;
         yddott;
         thetaddott];



function generatePlots(Tn, t, actualStates, desiredStates, error, u)
    close all; % close all opened figures
    % Tn = number of discrete time/path parameter points
    % t = time/path parameter history, dimension = Tn x 1
    % actualState = actual state of the system, dimension = Tn x n
    % desiredState = desired state of the system, dimension = Tn x n
    % error = desired state - actual state, dimension =  Tn x n
    % u = control inputs, dimension = Tn x m


    % Plot the robot's  velocities,
    figure
    subplot(2,1,1)
    plot(t,u(:,1), 'k-','LineWidth', 1.5);
    xlabel('Time [s]');
    ylabel('Linear speed [m/s]');
    grid on

    subplot(2,1,2)
    plot(t,u(:,2), 'k--','LineWidth', 1.5);
    xlabel('Time [s]');
    ylabel('Angular speed [rad/s]');
    grid on

    savefilename = ['OUT/controlInputs'];
    saveas(gcf, savefilename, 'fig');
    print('-depsc2', '-r300', [savefilename, '.eps']);

    % Create a movie of the simulation (path/trajectory)


    xmax = max(desiredStates(:,1));
    xmin = min(desiredStates(:,1));
    ymax = max(desiredStates(:,2));
    ymin = min(desiredStates(:,2));

    vid = VideoWriter('OUT/trajectory.avi');
    vid.Quality = 100;
    vid.FrameRate = 5;
```

```
    open(vid)

    fig=figure;
    clf reset;

    for i = 1:Tn,
        clf;
        box on;
        axis([xmin-5 xmax+5 ymin-5 ymax+5]);
        axis equal;
        axis manual;
        [Xa,Ya] = plot_DDMR(actualStates(i,:),axis(gca)); %
        hold on;
        desired =
plot(desiredStates(1:i,1),desiredStates(1:i,2),'LineWidth',1.5);
        hold on
        actual = plot(actualStates(1:i,1),actualStates(1:i,2),'k--');
        fill(Xa,Ya,'r');
        hold off;
        xlabel('x [m]');
        ylabel('y [m]');
        F = getframe(fig);
        writeVideo(vid,F);
    end
    [Xa,Ya] = plot_DDMR(actualStates(1,:),axis(gca)); % DDMR =>
Differential drive mobile robot
    grid on
    hold on
    plot(Xa,Ya);
    hold on
    %legend([actual desired],'actual', 'desired');
    savefilename = 'OUT/trajectory';
    saveas(gcf, savefilename, 'fig');
    print('-depsc2', '-r300', [savefilename, '.eps']);

    close(vid);

    % Create the movie and save it as an AVI file
    % winopen('OUT/trajectory.avi')



function [X,Y] = plot_DDMR(Q,AX)
% PLOT_UNICYCLE   Function that generates lines for plotting a unicycle.
```

```
%
%    PLOT_UNICYCLE(Q,AX) takes in the axis AX = axis(gca) and the unicycle
%    configuration Q and outputs lines (X,Y) for use, for example, as:
%       fill(X,Y,'b')
%    This must be done in the parent script file.

x     = Q(1);
y     = Q(2);
theta = Q(3);

l1 = 0.02*max([AX(2)-AX(1),AX(4)-AX(3)]);
X = [x,x+l1*cos(theta-2*pi/3),x+l1*cos(theta),x+l1*cos(theta+2*pi/3),x];
Y = [y,y+l1*sin(theta-2*pi/3),y+l1*sin(theta),y+l1*sin(theta+2*pi/3),y];
```

3.3 Division of Labor

Anthony Le - Helped write the code for Matlab and Pioneer, block diagram, put comments in the code

Jason Morris - Helped write the code for Pioneer, objective, background, implementation in lab report

Mike McGrath - Helped write the code for Pioneer, flow chart, table of contents, and Summary and Conclusion in lab report

Of course, throughout the lab all three of us consulted with each other and helped each other when necessary. Most of the time we were all working on the same thing at the same time. Everybody did roughly the same amount of work.

3.4 Summary and Conclusion

We were not able to implement leader-follower formation control using two Pioneer 3-DX robots (Part 2). We simply ran out of time, but we were able to simulate it. We implemented part 1 which is just the leader robot. We simulated our code on V-REP to debug the code and to make sure it worked before testing

on the actual robots. We also were able to control both robots wirelessly using MATLAB.

In summary, we learned how to use a leader-follower formation with a figure-eight pattern. We implemented this both by simulating them using Matlab and V-REP and also by testing them in the lab using the actual robot.