



ECE 467: Robotics Design Laboratory

Lab 4: 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)

2 Background

2.1 Software

2.2 Hardware

2.3 Block Diagram

3 Implementation

3.1 Steps

3.2 Implementation Code

3.3 Division of Labor

3.4 Summary and Conclusion

1 Objective(s)

In this lab, our goal was to learn formation control algorithms using linear approximation 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

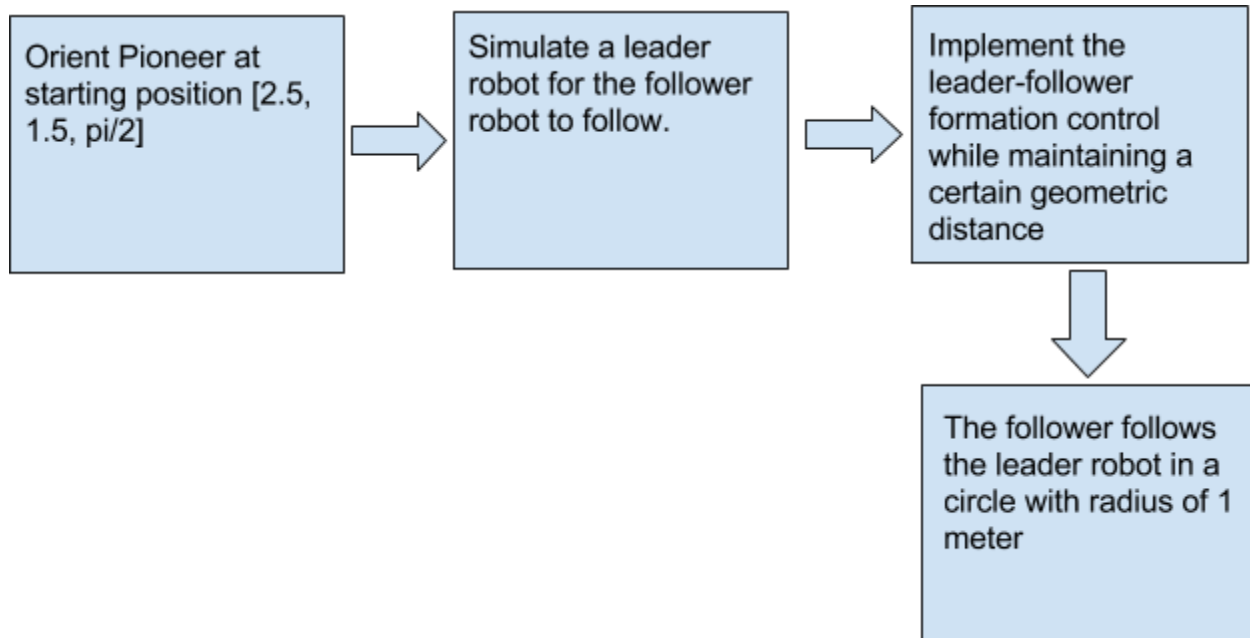
2.1 Software

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

2.3 Block Diagram



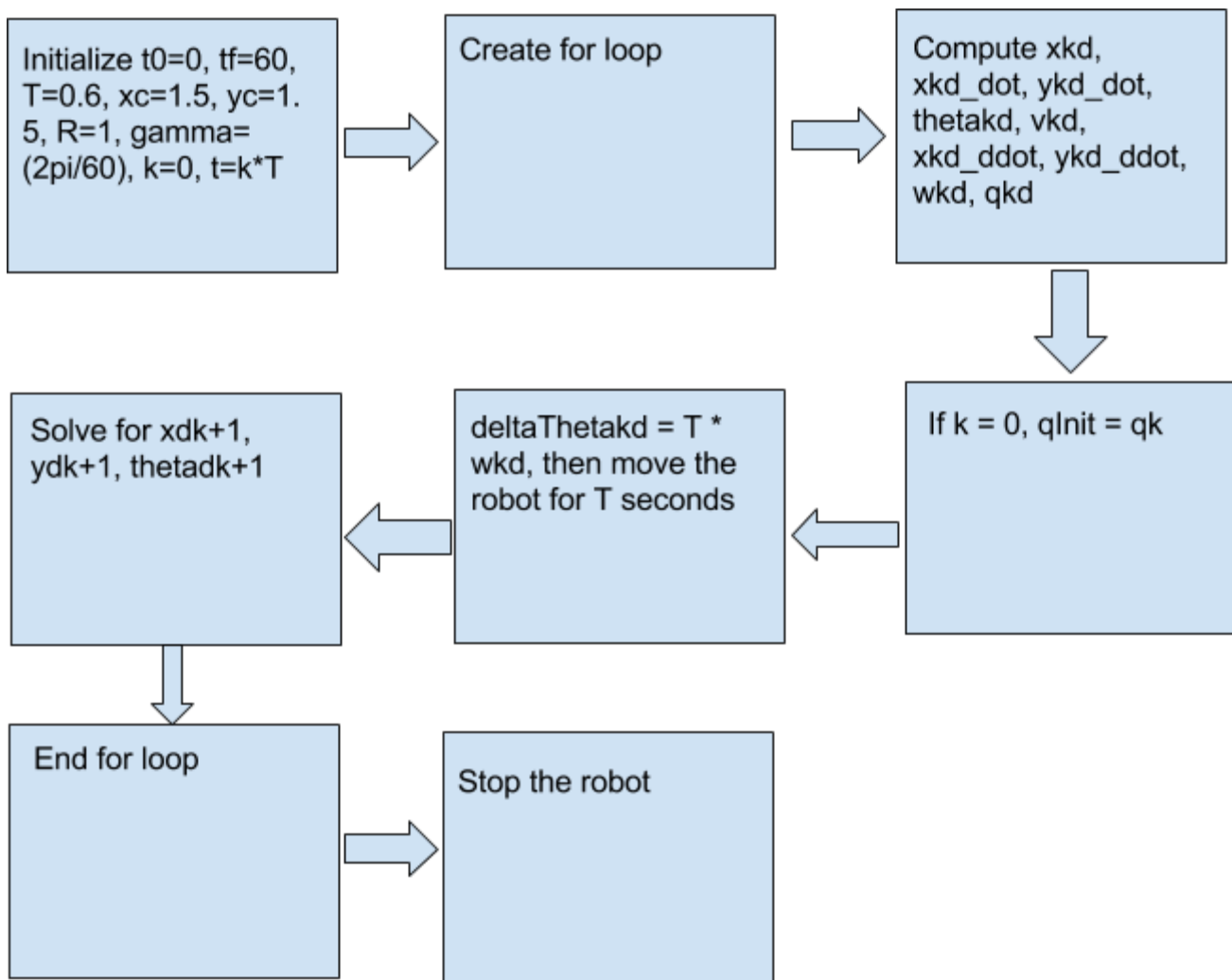
3 Implementation

3.1 Steps

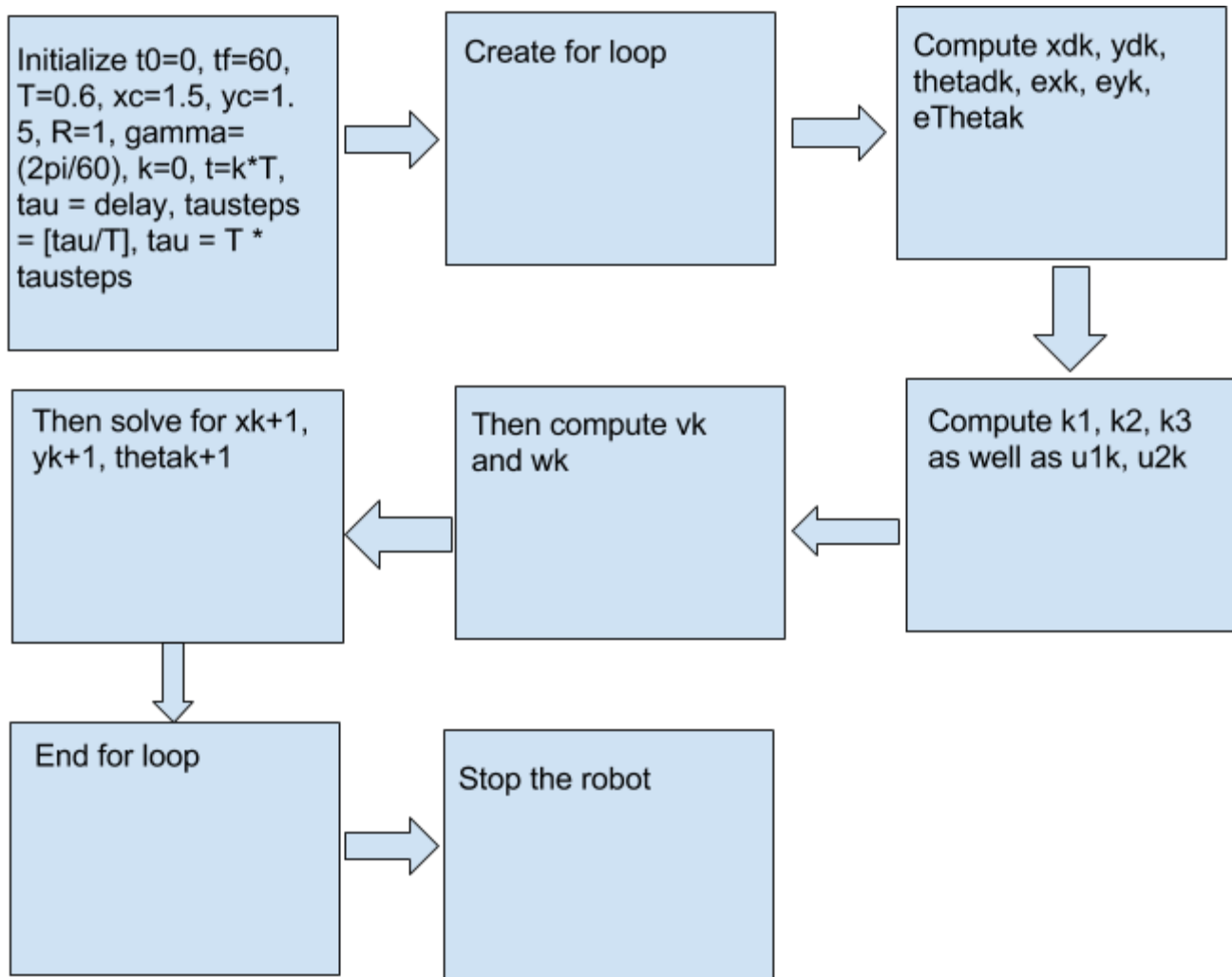
1. Used implementation steps given in class to create prelab code for part 1.
2. With example code given on Sakai, we were able to implement prelab code to wirelessly control the robot.
3. Simulated part 1 code on V-REP
4. Tweaked part 1 prelab code along with implementation steps given in class to create 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 = PrelabPart1
% 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 = 60; % initial and final simulation time [s]
T = 0.6; % 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 = 2*pi/(tf-t0);
% Generate reference trajectory (circle)
R = 1; % radius of the circle [m]
center = [1.5 1.5]; % [m]
for k = 0:1:tsteps
    t = k*T;
    xdt = center(1) + R*cos(gamma*t);
    ydt = center(2) + R*sin(gamma*t);

    xd_dott = -R*gamma*sin(gamma*t);
    yd_dott = R*gamma*cos(gamma*t);
    thetadt = atan2(yd_dott, xd_dott);

    xd_dot_dott = -R*gamma^2*cos(gamma*t);
    yd_dot_dott = -R*gamma^2*sin(gamma*t);

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

    vkd(k+1, 1) = sqrt(xd_dott.^2 + yd_dott.^2);

    qk = [xdt,ydt,thetadt];

    if k == 0
        desiredPose = qk;
    end
end

```

```
delta_theta = T*omegadt(1,1);
xdtnew = xdt + T*vkd(1,1)*cos(thetadt + (delta_theta/2));
ydtnew = ydt + T*vkd(1,1)*sin(thetadt + (delta_theta/2));
thetadtnew = thetadt + delta_theta;
qknew = [xdtnew,ydtnew,thetadtnew];

desiredPose(k+2, :) = qknew

end

desired = desiredPose;
generatePlots(length(dt), dt, [], desiredPose, [], [vkd 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 PrelabPart2
% 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 = 60; % initial and final simulation time [s]
T = 0.6; % 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 = 2*pi/(tf-t0);

Tau = 3;
Tausteps = (Tau/T);
Tau = T*(0:Tausteps)';

qInit = [2, 1, (pi/2)];
qkprnew = qInit;

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

zeta = 0.5;
a = 0.2;

% dtnew(1, 1) = 0;

desiredpose = PrelabPart1new();
for i = 0:4
desiredpose(102+i,:) = desiredpose(101,:);
end

for k = 0:tsteps+Tausteps

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

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

%         dtnew(kpr,1) = t;
%         actualPose(kpr, :) = qkprnew;

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

```

```

%      x = desiredpose(kpr,1) - (e_xpr.*cos(theta) -
e_ypr.*sin(theta));
%      y = desiredpose(kpr,2) - (e_xpr.*sin(theta) +
e_ypr.*cos(theta));

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

%      k1 = 2*zeta*a;
      k1 = 1.7;
      k2 = ((a.^2) - ((desiredpose(kpr, 4)).^2))/(abs(desiredpose(kpr,
5)))));
      k3 = k1;

      u1pr = -k1.*e_xpr;
      u2pr = -k2.*e_ypr - k3.*e_thetapr;

      vkpr(k+1, 1) = desiredpose(1,5)*cos(e_thetapr)-u1pr;
%      tmppr = yd_dot_dottpr.*xd_dottpr - xd_dot_dottpr.*yd_dottpr;
      omegadtpr(k+1, 1) = desiredpose(kpr, 4) - u2pr;

      delta_thetapr = T*omegadtpr(k+1,1);
%      xkprnew = x+T*vkpr(k+1,1)*cos((theta)+(delta_thetapr/2));
%      ykprnew = y+T*vkpr(k+1,1)*sin((theta)+(delta_thetapr/2));
%      xkprnew = desiredpose(k+1,
1)+T*vkpr(k+1,1)*cos((theta)+(delta_thetapr/2));
%      ykprnew = desiredpose(k+1,
2)+T*vkpr(k+1,1)*sin((theta)+(delta_thetapr/2));

%      try this next:
      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;

      qkprnew = [xkprnew,ykprnew,thetakprnew];
%      actualPose(k+2, :) = qkprnew;

      end
end

% disp((dtnew));

```

```

% disp(lengdesiredpose);
for i = 1:3
newDesired(:,i) = desiredpose(:,i);
end

% poseError = newDesired-actualPose;
% disp(length(actualPose));
% disp(length(desiredpose));

disp(length(actualPose));
disp(length(desiredpose));

poseError = newDesired-actualPose;
% generatePlots(length(dtnew), dtnew, actualPose, desiredpose, [], [vdt
omegadot]);
% generatePlots(length(dtnew), dtnew, actualPose, newDesired, poseError,
[desiredpose(:, 5) desiredpose(:, 4)]);
generatePlots(length(dtnew), dtnew, actualPose, desiredpose, poseError,
[desiredpose(:, 5) desiredpose(:, 4)]);

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-4 xmax+5 ymin-4 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];

```

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 able to implement leader-follower formation control using two Pioneer 3-DX robots. We simulated our code on V-REP to make 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 circle trajectory. We implemented theis both by simulating them using Matlab and V-REP and also by testing them in the lab using the actual robot.