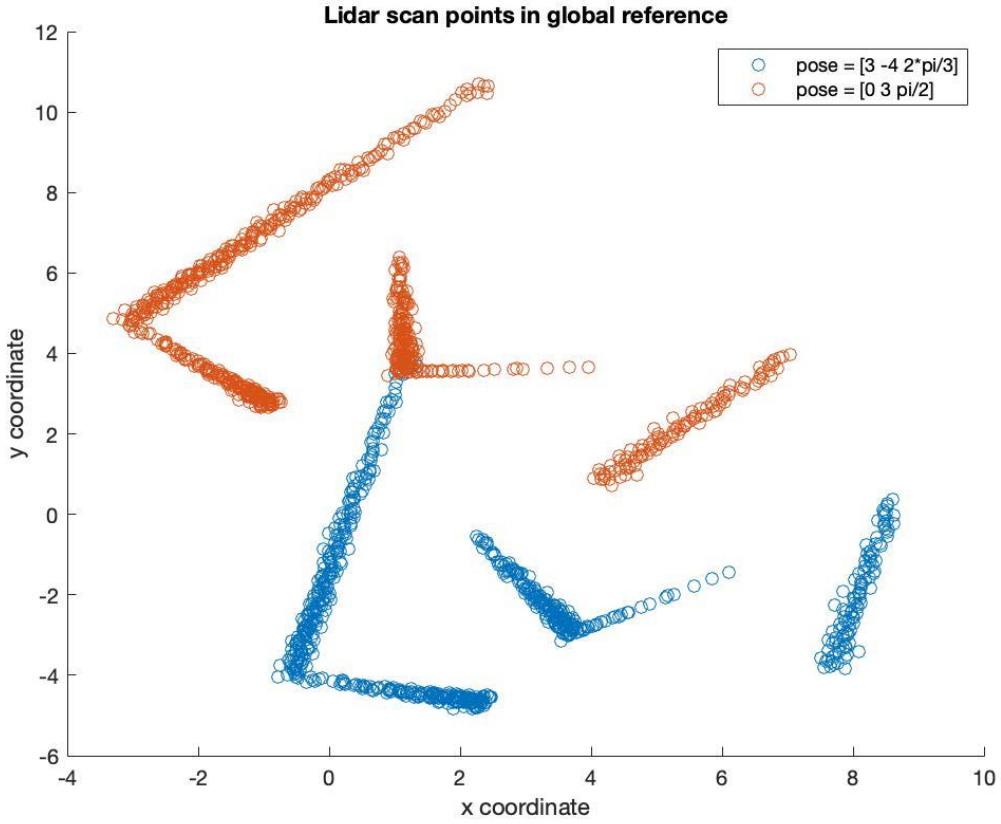
Jonathan Nusantara (jan265)
ECE 5772
HW 1

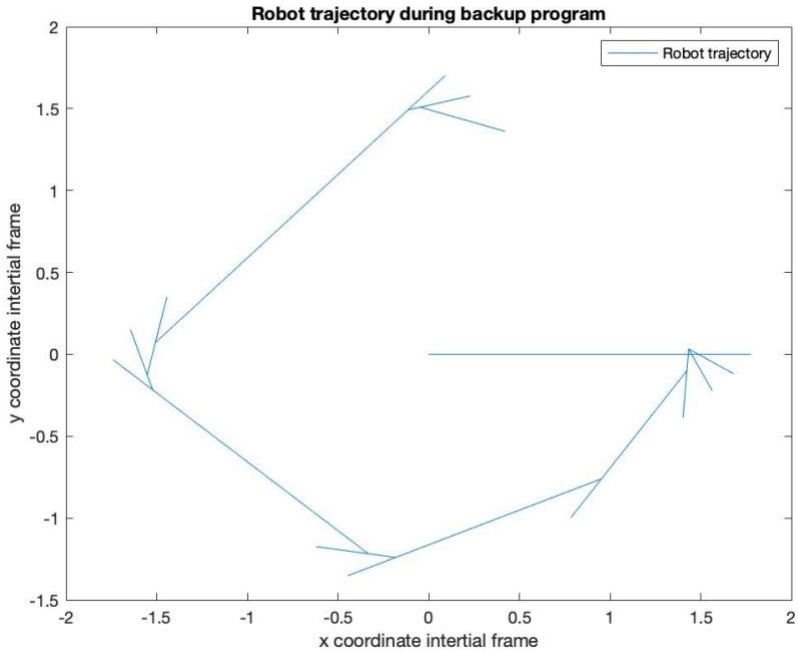
Changing reference frames:

1. Function submitted to canvas.
2. Function submitted to canvas.
3. Function submitted to canvas.
4. File called plot_lidar.m is created.

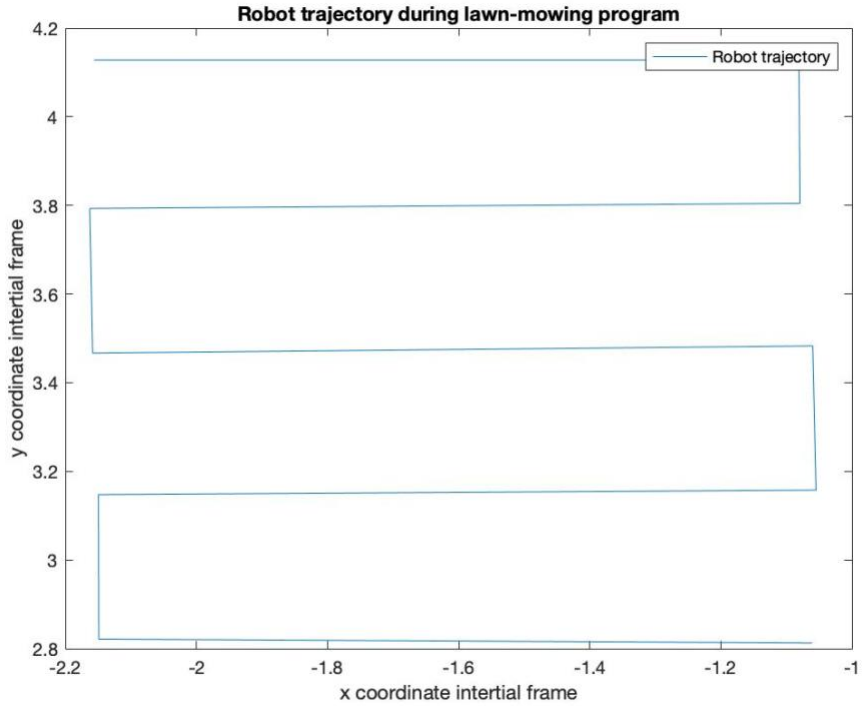


Getting to know the simulator and running a simple control program:

1. setFwdVelAngVelCreate: This is a function/command used to set the forward and angular velocity of the robot. It allows us to move or rotate the robot. The inputs are Robot (port configuration and robot's name), scaled forward velocity in m/s and scaled angular velocity in rad/s.
2. If the wheel velocities are more than 0.5m/s, the motor wheels will saturate and unwanted behavior will happen. Function submitted to canvas.
3. turnInPlace.m edited.
4. The file created is called backupBump.m. The plot is called plot_trajectory_backup_function.fig

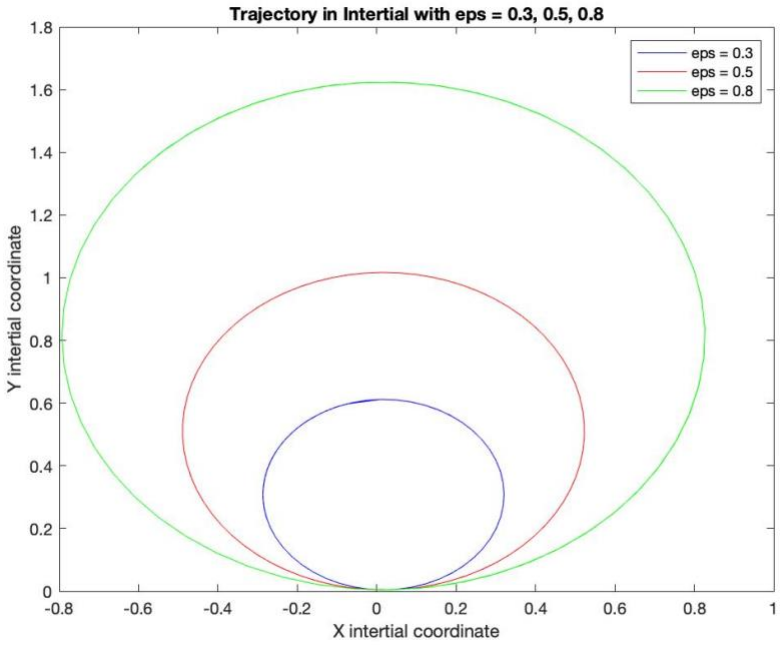


5. The control function that I created is called lawnMowing.m. The goal is to have the robot move like a lawnmower. The robot will go to one direction for 1 meter, then turn 90 deg clockwise, then move forward for 0.25m, then turn 90deg clockwise again, then to proceed moving forward for another meter before turning 90deg anti-clockwise, moving forward 0.25m, and turning 90deg anti-clockwise again. It will then continuously repeat it. The trajectory plot plot_trajectory_lawnmowing.fig is as below:



Feedback Linearization:

1. Submitted to canvas.
2. The epsilon value of 0.3, 0.5, and 0.8 are plotted. The code used is localControl.m and plot_localControl.m.



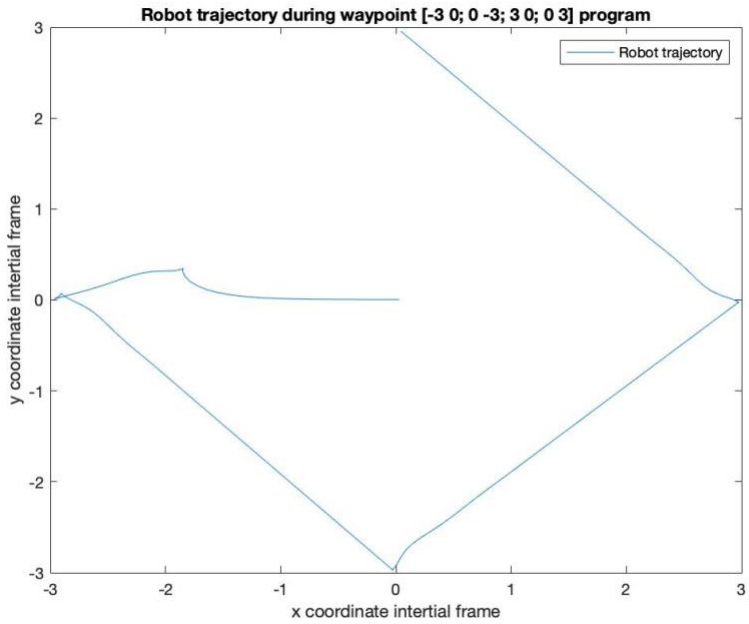
3. The function visitWaypoints.m has been created and included in the zip file.

```
      function[dataStore] = visitWaypoints(Robot,maxTime)
% TURNINPLACE: simple example program to use with iRobot Create (or
simulator).
% Reads data from sensors, makes the robot drive through the waypoints, and
saves a datalog.
%
%   dataStore = backupBump(Robot,maxTime) runs
%
%   INPUTStype
%       Robot       Port configurations and robot name (get from running
CreatePiInit)
%       maxTime     max time to run program (in seconds)
%
%   OUTPUTS
%       dataStore   struct containing logged data


%
%   NOTE: Assume differential-drive robot whose wheels turn at a constant
%         rate between sensor readings.
%
%   Cornell University
%   MAE 5180: Autonomous Mobile Robots
%   Homework #1
```

```
%   Nusantara, Jonathan

% Set unspecified inputs
if nargin < 1
    disp('ERROR: TCP/IP port object not provided.');
    return;
elseif nargin < 2
    maxTime = 100;
end

try
    % When running with the real robot, we need to define the appropriate
    % ports. This will fail when NOT connected to a physical robot
    CreatePort=Robot.CreatePort;
catch
    % If not real robot, then we are using the simulator object
    CreatePort = Robot;
end

% declare dataStore as a global variable so it can be accessed from the
% workspace even if the program is stopped
global dataStore;

% initialize datalog struct (customize according to needs)
dataStore = struct('truthPose', [],...
                   'odometry', [], ...
                   'rsdepth', [], ...
                   'bump', [], ...
                   'beacon', []);


% Variable used to keep track of whether the overhead localization "lost"
% the robot (number of consecutive times the robot was not identified).
% If the robot doesn't know where it is we want to stop it for
% safety reasons.
noRobotCount = 0;

% Initialization of local variable
counter = 1 % Loop counter
epsilon = 0.2;
gotopt = 1;
closeEnough = 0.1; % Acceptable robot distance from waypoint
waypoints = [-1 0; 1 0 ]; % List of points for the robot to track
waypoints_count = length(waypoints);

% Robot pose initialization
[noRobotCount,dataStore]=readStoreSensorData(Robot,noRobotCount,dataStore);
lastX = dataStore.truthPose(counter, 2);
lastY = dataStore.truthPose(counter, 3);
lastTheta = dataStore.truthPose(counter, 4);



SetFwdVelAngVelCreate(CreatePort, 0,0);
tic
```

```
while toc < maxTime && gotopt <= waypoints_count

    % READ & STORE SENSOR DATA

[noRobotCount,dataStore]=readStoreSensorData(Robot,noRobotCount,dataStore);

    % CONTROL FUNCTION (send robot commands)

    % Set velocity
    %cmdV = 0.3;
    %cmdW = 0;

    % if overhead localization loses the robot for too long, stop it
    if noRobotCount >= 3
        SetFwdVelAngVelCreate(CreatePort, 0,0);

    else
        % Calculate the required V and W based on current robot pose and
        % waypoint
        [cmdV,cmdW] = feedbackLin(waypoints(gotopt,1) -
dataStore.truthPose(counter, 2),waypoints(gotopt,2) -
dataStore.truthPose(counter, 3),dataStore.truthPose(counter, 4),epsilon);
        [cmdV,cmdW] = limitCmds(cmdV, cmdW, 0.49, 0.13);
        SetFwdVelAngVelCreate(CreatePort, cmdV, cmdW );

        % If robot is within a range to current waypoint, go to next point
        if (sqrt((dataStore.truthPose(counter, 2) - waypoints(gotopt,1)).^2 +
(dataStore.truthPose(counter, 3) - waypoints(gotopt,2)).^2)) < closeEnough;
            gotopt = gotopt + 1
        end

    end

    pause(0.1);
    counter = counter + 1; % Update loop counter
end

% set forward and angular velocity to zero (stop robot) before exiting the
function
SetFwdVelAngVelCreate(CreatePort, 0,0 );

% Plot the trajectory
figure
plot(dataStore.truthPose(:,2),dataStore.truthPose(:,3))
legend('Robot trajectory')
title('Robot trajectory during waypoint [-1 0; 1 0 ] program')
xlabel('x coordinate intertial frame')
ylabel('y coordinate intertial frame')
savefig('plot_trajectory_waypoint2.fig')
```

4. For the experiment below, the robot started at x=0 and y=0 inertial frame. The first trajectory for [-3 0; 0 -3; 3 0; 0 3] called plot_trajectory_waypoint1.fig is:



The second trajectory for [-1 0; 1 0] called plot_trajectory_waypoint2.fig is: