Astana IT University
Smart Technologies

Dead-reckoning localization and motion control for differential-drive mobile robot

Introduction to Intelligent Systems

ST-2102

Profesor: Kusdavletov Sanzhar

Assemgul Saparova

Introduction

Mobile robot is a device that uses sensors for environment identification and which is controlled by a software. Its architecture is a blend of artificial intelligence (AI) and physical hardware (wheels, tracks, legs) [1]. Based on the environment in which the mobile robot operates, it can be classified into different categories:

- **Polar robots** that navigate icy and uneven surfaces.
- **Aerial robots**, which float in the air.
- **Land/home robots** that travel on dry land or homes.
- **Underwater robots**, which are used underwater
- **Delivery and transportation robot** that are designed to transport items.

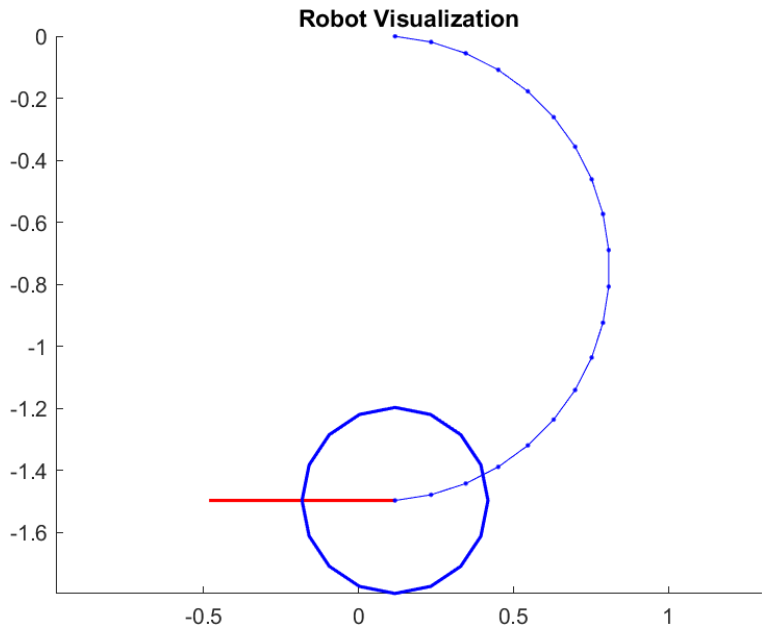
Furthermore, there are two challenges that are considered with the design of mobile robots: localization and motion control. In this report, I will present solutions for these challenges and share my results.

Differential Drive Mobile Robot

Firstly, before proposing a solution for any of the problems, I will initialize a Differential Mobile Robot and visualize it in a two-dimensional space:

```
wheelRadius = 0.1;
wheelBase = 0.5;
% initialize a differential drive robot
dd = DifferentialDrive(wheelRadius, wheelBase)

viz = Visualizer2D;
startLoc = [0, 0, 0]; % starting position of a robot
viz.robotRadius = 0.3;

sampleTime = 0.5;
tVec = 0:sampleTime:10; % time array
r= rateControl(1/sampleTime); % rate control

% compute velocity (v) and angular velocity (w)
wL = pi;
wR = pi/2;
[v,w] = forwardKinematics(dd,wL,wR);

bodyV = [v;0;w];

currentLoc = startLoc;

for idx = 2:numel(tVec)
```

```
        % convert from robot body to world
        vel =  bodyToWorld(bodyV,currentLoc);

        % update robot's current location
        currentLoc = currentLoc + vel * sampleTime;

        % update its location in a 2d space
        viz(currentLoc);

        waitfor(r);
    end
```

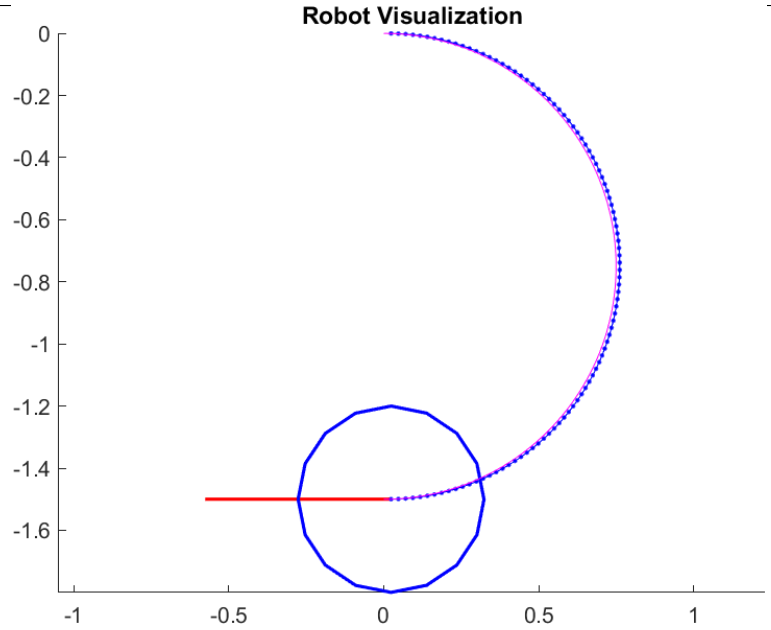


Localization

Robot localization is a problem that deals with the robot's current location and determining where it is located with respect to its environment [2]. Overall, the information used to compute the robot's current location may be gathered from various sensors that are mounted on the robot. However, the errors that are presented in those sensor measurements makes the obtained pose unreliable.
In this section, I will implement a solution for the localization problem with the use of odometer sensors. They can estimate the change in position over a specified time. Moreover, for a differential drive robot, it is easy to compute the robot's location when we know its wheels radio, distance from the center of the robot to each wheel, and information that we gathered from the odometer.

```
wheelRadius = 0.1;
wheelBase = 0.5;
% initialize a differential drive robot
dd = DifferentialDrive(wheelRadius, wheelBase);
```

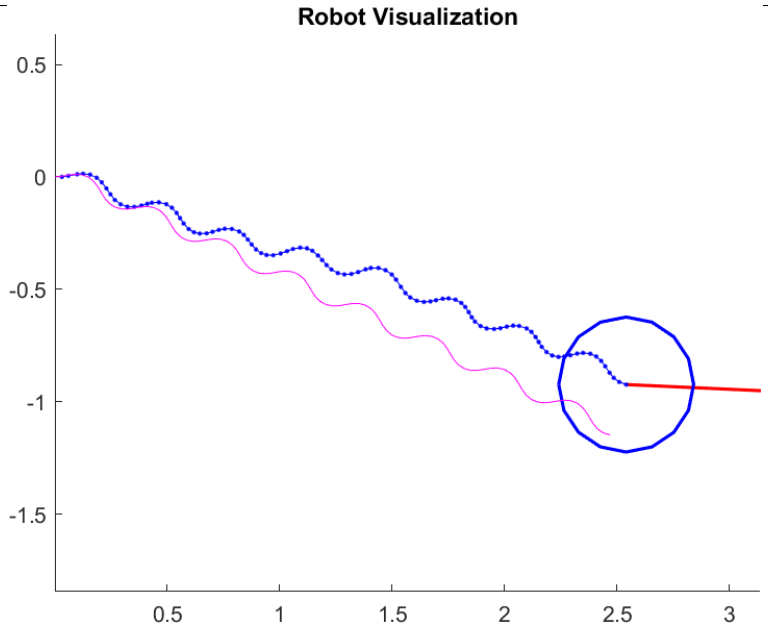
```
viz = Visualizer2D;
startLoc = [0, 0, 0]; % starting position of a robot
viz.robotRadius = 0.3;

% compute velocity (v) and angular velocity (w)
wL = pi;
wR = pi/2;
[v,w] = forwardKinematics(dd,wL,wR);

bodyV = [v;0;w];

[sampleTime,tVec,r] = simulTime(0.1,10);

% array of odometry poses
odom_pose = zeros(3,numel(tVec));
odom_pose(:,1) = startLoc;
currentLoc = startLoc;

for idx = 2:numel(tVec)
    % compute odometry
    d_sr = wheelRadius * wR * sampleTime;
    d_sl = wheelRadius * wL * sampleTime;
    d_s = (d_sr + d_sl)/2;
    d_theta = (d_sr - d_sl)/wheelBase;

    % store calculated odometries
    odom_pose(:,idx) = odom_pose(:,idx-1) + [ d_s *cos(odom_pose(3,idx-1) +  d_theta/2 );
                                              d_s *sin(odom_pose(3,idx-1) +  d_theta/2 );
                                              d_theta];

    % convert from robot body to world
    vel =  bodyToWorld(bodyV,currentLoc);

    % update robot's current location
    currentLoc = currentLoc + vel * sampleTime;

    % update its location in a 2d space
    viz(currentLoc);

    waitfor(r);
end

hold on
plot(odom_pose(1,:),odom_pose(2,:),'m');
```



However, due to many disturbance factors, such as wind, tire size, and uneven surface, the odometry may work correctly only in short time frames. That's why I will additionally implement a source of noise to make the simulation more realistic.

```
wheelRadius = 0.1;
wheelBase = 0.5;
% initialize a differential drive robot
dd = DifferentialDrive(wheelRadius, wheelBase);

viz = Visualizer2D;
startLoc = [0, 0, 0]; % starting position of a robot
viz.robotRadius = 0.3;

% compute velocity (v) and angular velocity (w)
wL = pi;
wR = pi/2;
[v,w] = forwardKinematics(dd,wL,wR);

bodyV = [v;0;w];

varianceV = 0.03;
varianceW = 0.13;

[sampleTime,tVec,r] = simulTime(0.1,10);

% array of odometry poses
odom_pose = zeros(3,numel(tVec));
odom_pose(:,1) = startLoc;
currentLoc = startLoc;

for idx = 2:numel(tVec)
    v = 0.3; % const linear velocity
```

```
        w = pi*(cos(idx/2)); % angular velocity that will change
        [wL,wR] = inverseKinematics(dd,v ,w);

        % add noise to the velocities
        noiseV = v  + normrnd(0,varianceV);
        noiseW = w + normrnd(0,varianceW);
        bodyV = [noiseV ;0;noiseW];

        % compute odometry
        d_sr = wheelRadius * wR * sampleTime;
        d_sl = wheelRadius * wL * sampleTime;
        d_s = (d_sr + d_sl)/2;
        d_theta = (d_sr - d_sl)/wheelBase;

        % store calculated odometries
        odom_pose(:,idx) = odom_pose(:,idx-1) + [ d_s *cos(odom_pose(3,idx-1) +  d_theta/2 );
                                                  d_s *sin(odom_pose(3,idx-1) +  d_theta/2 );
                                                  d_theta];

        % convert from robot body to world
        vel =  bodyToWorld(bodyV,currentLoc);

        % update robot's current location
        currentLoc = currentLoc + vel * sampleTime;

        % update its location in a 2d space
        viz(currentLoc);

        waitfor(r);
    end

    hold on
    plot(odom_pose(1,:),odom_pose(2,:),'m');
```



From the figure above, the pink line represents the odometry computation and the blue line represents the robot's trajectory. These lines differ, because odometry is computed from velocities without noise, and the robot's trajectory represents its *real* trajectory (with noise).

Motion Control

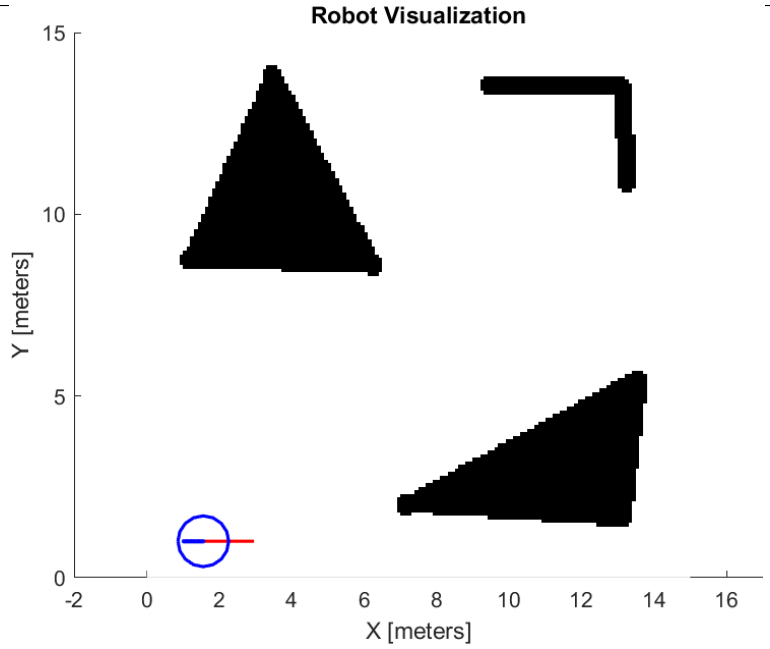
In robotics, motion control is the process of moving the machine using rotary and linear actuators [3]. Motion control can be applied in many ways, for example, in simulating missile flights, wind picturing, etc. In this section, I will be concerned with implementing solutions for the obstacle avoidance and path tracking problems of the motion control.

Obstacle Avoidance

There are numerous solutions for obstacle avoidance in mobile robotics, one of which includes the application of Laser Imaging Detection and Ranging (LIDAR) sensor. LIDAR is a sensor that emits infrared laser lights to measure distance from the reference point to objects. The distance of an object from the sensor is calculated based on the return time [4]. I will firstly start with attaching a LIDAR to my differential drive robot and setting up a world environment for it.

```
lidar = LidarSensor;
lidar.sensorOffset = [0, 0];
% number of sensors and their angle
lidar.scanAngles = linspace(-pi/2, pi/2, 51);
lidar.maxRange = 0.7;

% load an environment map
map = generateMap('maps\map1.png', 0.1, 10);

wheelRadius = 0.3;
wheelBase = 0.5;
```

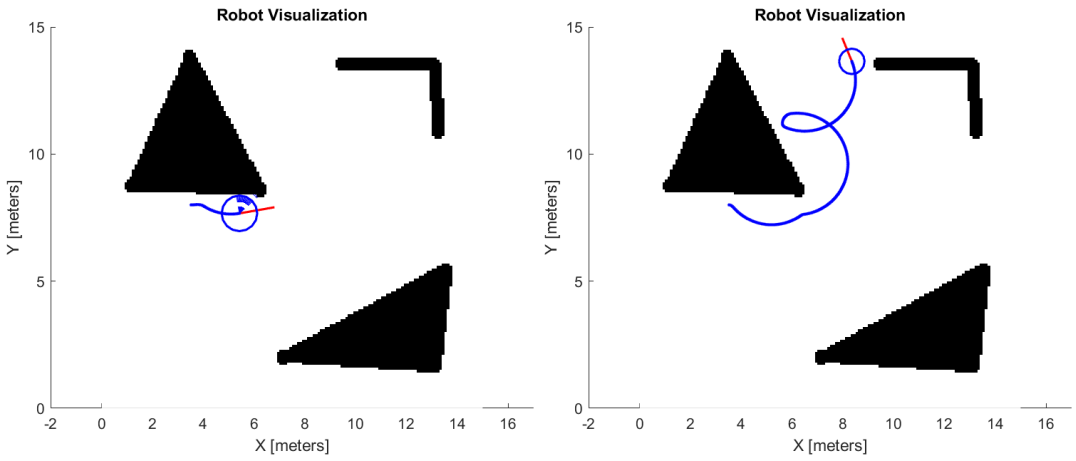
```
% initialize a differential drive robot
dd = DifferentialDrive(wheelRadius, wheelBase);

viz = Visualizer2D;
startLoc = [1; 1; 0]; % starting position of a robot
viz.robotRadius = 0.7;
% assign the map to the visual space
viz.mapName = 'map';

% attach lidar sensor to the robot
attachLidarSensor(viz, lidar);

v = 0.2;
w = 0;
bodyV = [v; 0; w];

currentLoc = startLoc;

[sampleTime,tVec,r] = simulTime(0.1,10);

for idx = 2:numel(tVec)
    % convert from robot body to world
    vel =  bodyToWorld(bodyV,currentLoc);

    % update robot's current location
    currentLoc = currentLoc + vel * sampleTime;

    % obtain sensor data
    ranges = lidar(currentLoc);

    % update its location in a 2d space
    viz(currentLoc, ranges);

    waitfor(r);
end
```



I can further make use of the LIDAR sensor to help the robot to avoid any obstacles. I will use Vector Field Histogram (VFH) that will use LIDAR data.

```
lidar = LidarSensor;
lidar.sensorOffset = [0, 0];
% number of sensors and their angle
lidar.scanAngles = linspace(-pi/2, pi/2, 51);
lidar.maxRange = 1;

% load an environment map
map = generateMap('maps\map1.png', 0.1, 10);

wheelRadius = 0.3;
wheelBase = 0.5;
% initialize a differential drive robot
dd = DifferentialDrive(wheelRadius, wheelBase);

viz = Visualizer2D;
startLoc = [3.5; 8; 0]; % starting position of a robot
viz.robotRadius = 0.5;
% assign the map to the visual space
viz.mapName = 'map';

% attach lidar sensor to the robot
attachLidarSensor(viz, lidar);

v = 0.2;
w = 0;
bodyV = [v; 0; w];

vfh = controllerVFH;
```

```
vfh.DistanceLimits=[0.35 2];
vfh.RobotRadius = viz.robotRadius;
vfh.SafetyDistance = .3;
vfh.MinTurningRadius = .5;

currentLoc = startLoc;

[sampleTime,tVec,r] = simulTime(0.1,75);

for idx = 2:numel(tVec)
    % obtain sensor data
    ranges = lidar(currentLoc);
    targetDir = 0.1;
    referenceW = vfh(ranges, lidar.scanAngles, targetDir);
    if isnan(referenceW)
        referenceW = 0.5;
    end

    bodyV = [v;0;referenceW];

    % convert from robot body to world
    vel =  bodyToWorld(bodyV,currentLoc);

    % update robot's current location
    currentLoc = currentLoc + vel * sampleTime;

    % update its location in a 2d space
    viz(currentLoc, ranges);

    waitfor(r);
end
```



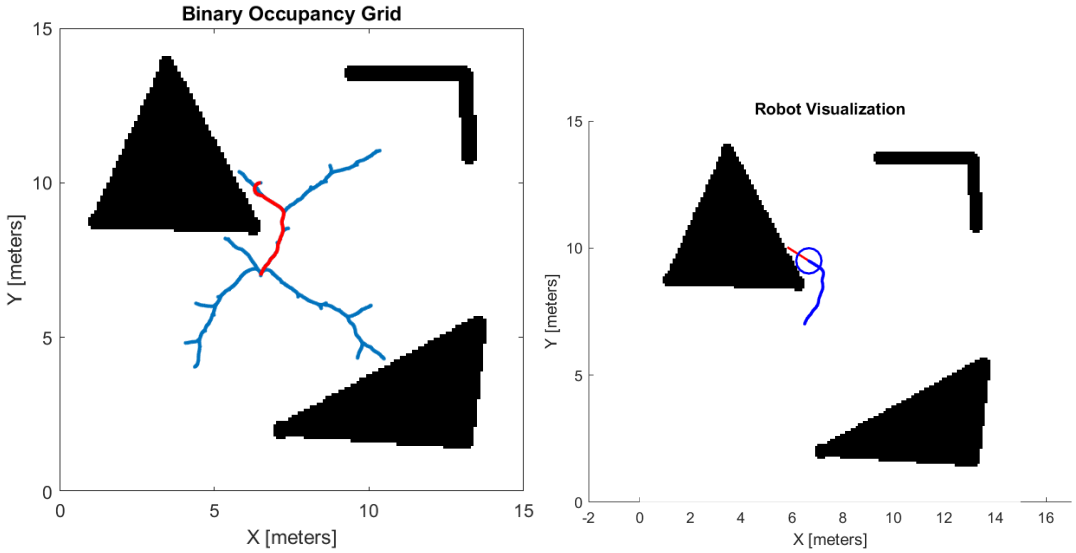
As can be seen from the figure above, the LIDAR detects an obstacle using the infrared lights and the robot turns away from the obstacle and continues its way until it finds another obstacle.

Path Tracking

This section is concerned with making the robot go to an explicit point in the environment using a Pure-Pursuit algorithm. Overall, PurePursuit is a path tracking algorithm, which moves the robot from its starting position to a specified position by computing the angular velocity command [5].

```
startLoc = [6.5; 7; 1.5]; % starting position of a robot
goalLoc = [6.5, 10,0]; % desired location to go to

% load an environment map
map = generateMap('maps\map1.png', 0.1, 10);
% process the map
boubds = [map.XWorldLimits; map.YWorldLimits; [-pi pi]];
statespace = stateSpaceDubins(boubds);
statespace.MinTurningRadius = 0.2;
image = imread('maps\map1.png');
grayscale = rgb2gray(image);
bwimage = grayscale < .1;

% morphological dilation
se = strel('sphere',2);
bwimage = imdilate(bwimage,se);

% convert the map to  binaryOccupancyMap
mapDilatation = binaryOccupancyMap(bwimage,10);

show(mapDilatation)

% calculate the path for the robot
stateValidator = validatorOccupancyMap(statespace);
stateValidator.Map = mapDilatation;
% check for an obstacle between this distance in the path
stateValidator.ValidationDistance = .1;
planner = plannerRRTStar(statespace, stateValidator);
planner.MaxConnectionDistance = .02;
planner.MaxIterations = 30000;
planner.GoalReachedFcn = @exampleHelperCheckIfGoal;
rng(1,'twister')
[pthObj, solnInfo] = plan(planner, startLoc.', goalLoc);

show(map)
hold on
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-');
% plot path
interpolate(pthObj,300)
plot(pthObj.States(:,1), pthObj.States(:,2), 'r-', 'LineWidth', 2)
hold off

wheelRadius = 0.3;
wheelBase = 0.5;
% initialize a differential drive robot
dd = DifferentialDrive(wheelRadius, wheelBase);
```

```
viz = Visualizer2D;
viz.robotRadius = 0.5;
% assign the map to the visual space
viz.mapName = 'map';

% PurePursuit for path tracking
controller = controllerPurePursuit;
% poses calculate by RRT algorithm
controller.Waypoints = [pthObj.States(:,1), pthObj.States(:,2) ];
% how far along the route the robot should look to find angular velocity
controller.LookaheadDistance = .05;
controller.DesiredLinearVelocity = 0.1;
controller.MaxAngularVelocity = pi/6;

currentLoc = startLoc;

[sampleTime,tVec,r] = simulTime(0.1,30);

for idx = 2:numel(tVec)
    [referenceV,referenceW,lookAheadPt] = controller(currentLoc);
    bodyV = [referenceV;0;referenceW];

    % convert from robot body to world
    vel =  bodyToWorld(bodyV,currentLoc);

    % update robot's current location
    currentLoc = currentLoc + vel * sampleTime;

    % update its location in a 2d space
    viz(currentLoc);

    waitfor(r);
end
```



Bibliography

[1] What is a mobile robot? Definition from WhatIs.com. — techtarget.com. — https://www.techtarget.com/iotagenda/definition/mobile-robot-mobile-robotics#:~:text=A%20mobile%20robot%20is%20a,as%20wheels%2C%20tracks%20and%20legs.

[2] *Huang, Shoudong*. Robot Localization: An Introduction / Shoudong Huang, Gamini Dissanayake // Wiley Encyclopedia of Electrical and Electronics Engineering. — John Wiley Sons, Ltd, 2016. — Pp. 1–10. https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W8318.

[3] What is Motion Control? — Heason — heason.com. — https://www.heason.com/news-media/technical-blog-archive/what-is-motion-control-.

[4] What is lidar? Learn How Lidar Works — Velodyne Lidar — velodynelidar.com. — https://velodynelidar.com/what-is-lidar/.

[5] Pure Pursuit Controller - MATLAB & Simulink — mathworks.com. — https://www.mathworks.com/help/nav/ug/pure-pursuit-controller.html.