# Robotic Systems I: Homework I

**Instructor:** Konstantinos Chatzilygeroudis (costashatz@upatras.gr)

April 4 2024

## Introduction

The homework concerns searching for a path (with actions) that solves a specific problem for a differential drive mobile robot (Fig. 1). The homework is split into three parts/sub-questions: 1) modeling of the system, 2) implementation of an RRT path planner with actions, and 3) solving a specific path planning problem with the developed RRT algorithm and the differential drive mobile robot.

## 1 Modeling (20%)

For this homework we are going to use a differential drive mobile robot . The system can be seen in Fig. 1. The system "lives" in the 2D plane, and has two control inputs (velocities of the two wheels!).
The state of the system is as follows:

$$\boldsymbol{x} = \begin{bmatrix} \theta \\ x \\ y \end{bmatrix} \in \mathbb{R}^3$$

And the control inputs:

$$\boldsymbol{u} = \begin{bmatrix} u_L \\ u_R \end{bmatrix} \in \mathbb{R}^2$$

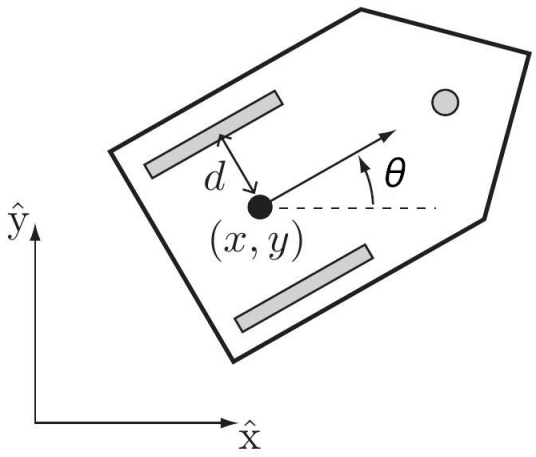


Figure 1: Visualization of the described system. Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

The continuous kinematics of the system are given by the following equations:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\theta} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\frac{r}{2d} & \frac{r}{2d} \\ \frac{r}{2}\cos\theta & \frac{r}{2}\cos\theta \\ \frac{r}{2}\sin\theta & \frac{r}{2}\sin\theta \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix}$$

where $(\theta, x, y)$ is the pose of the robot in world space, $r$ are the wheel radii, $d$ is the distance of the wheels from the robot center, and $u_L, u_R$ are the wheel angular velocities (commands).

In this part, you are asked to:

1. Write down a function in Python called `diffkin()` that takes as input the state $\boldsymbol{x} \in \mathbb{R}^{3\times 1}$ and controls $\boldsymbol{u} \in \mathbb{R}^{2\times 1}$ and returns the $\dot{\boldsymbol{x}} \in \mathbb{R}^{3\times 1}$.
2. Create a simple visualization of the system (e.g. by using the `matplotlib` library); you are free to use any library for the visualization
3. Simulate the system using Runge-Kutta 4th Order integration; create a simulation loop and verify that it works

Use the following values: $d = 0.25\,m, r = 0.1\,m, dt = 0.1\,s$. Also, we do not allow wheel velocities $(u_L, u_R)$ bigger than $0.5\,rad/s$.

For all the above, you are allowed to use only native Python functions/classes, the `numpy` library and some visualization library (e.g. `matplotlib`).

# 2 Rapidly exploring Random Trees (RRT) (50%)

## 2.1 RRT

**RRT Algorithm:**

1 : Initialize search tree with $\boldsymbol{x}_{\text{start}}$
2 : while *some stopping criteria is not met*
3 : sample $\boldsymbol{x}_{\text{sample}} \sim \mathcal{X}$
4 : find $\boldsymbol{x}_{\text{nearest}}$ nearest node of $\boldsymbol{x}_{\text{sample}}$ in tree
5 : connect $\boldsymbol{x}_{\text{nearest}}$ to $\boldsymbol{x}_{\text{new}}$ in direction of $\boldsymbol{x}_{\text{sample}}$
6 : **if** success
7 : add $\boldsymbol{x}_{\text{new}}$ to the tree with an edge from $\boldsymbol{x}_{\text{nearest}}$
8 : **if** $\boldsymbol{x}_{\text{new}} \in \mathcal{X}_{\text{goal}}$ return SUCCESS
9 : return FAILURE

## 2.2 Task

Your task is to implement generic code for a *kinodynamic* RRT algorithm. By *kinodynamic* RRT we mean the following:

- The **connect** function does not just draw a "line" between $\boldsymbol{x}_{\text{nearest}}$ and $\boldsymbol{x}_{\text{sample}}$ but *searches for a sequence of control inputs* that can get me as close as possible to $\boldsymbol{x}_{\text{sample}}$
- Usually this is implemented via an optimization routine
- You are free to implement the **connect** function with any optimization/hybrid way you wish, but the most straightforward (and slow) implementation is the following:
  - Sample $K$ random sequences of control inputs ($K$ is a "big" number)
  - Perform forward simulation for each sequence
  - Keep the best valid trajectory

In essence, we need a code that accepts any system with forward dynamics/kinematics, an initial and a target state and solves the problem.

For all the above, you are allowed to use only native Python functions/classes, the `numpy` library and some visualization library (e.g. `matplotlib`).

# 3 Differential Drive Robot Path Following (30%)

We now want to take the differential drive robot model from Sec. 1, and use the kinodynamic RRT algorithm developed in Sec. 2 to solve a path planning problem. You have to:

- Define the following task: *"The robot starts at* $(\theta_0, x_0, y_0)$ *and ends up at* $(\theta_1, x_1, y_1)$*.* $(x_0, y_0)$ *is at least 2 meters away from* $(x_1, y_1)$ *and* $\theta_0$ *is at least* $\frac{\pi}{2}$ *away from* $\theta_1$*".*
- Solve the above task using your developed RRT algorithm! Be careful with angle differences!
- Play with different initial and final positions and orientations
- Describe what you observe

For all the above, you are allowed to use only native Python functions/classes, the `numpy` library and some visualization library (e.g. `matplotlib`).

# 4 Deliverables

- Python file(s) with commented code[1]
- A short report

# Bonus Points

There is a 5% bonus if you include obstacles in your RRT algorithm.

[1] Jupyter notebooks are accepted as well.