# Homework 1: Safety Analysis

In this homework, we will focus on computing backward reachable tubes (BRTs) and safe sets for several dynamical systems. For programming, you are welcome to choose among a variety of computational tools, including the Level Set Toolbox and `helperOC` (MATLAB), BEACLS (C++), `DeepReach` (Python), `OptimizedDP` (Python) `JuliaReach` (Julia), etc. However, **I recommend using the code provided in `hw1_code.zip` which is written in MATLAB** and compatible with helperOC and Level Set Toolbox.

**Setup Instructions.**

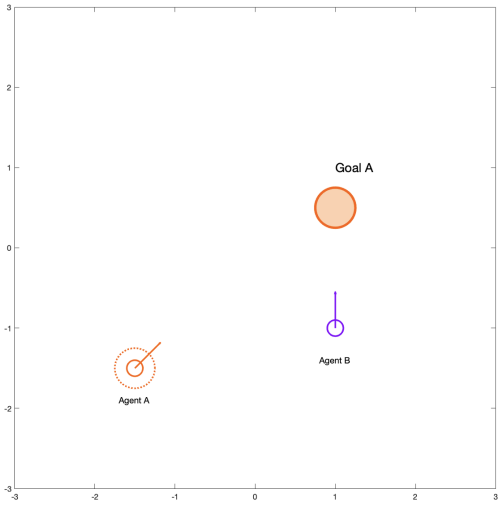
- If you don't have it already, install MATLAB through the university here: https://www.mathworks.com/academia/tah-portal/carnegie-mellon-university-544078.html.
- When installing MATLAB, also download the Image Processing Toolbox (needed for visualization) and Optimization Toolbox (needed for planning). If you already have MATLAB but aren't sure if you have this toolbox, just start running the code and it will tell you if this is an error. Then you can click on a link and download it on the spot.
- Extract `hw1_code.zip` into your favorite place.
- After loading MATLAB, make sure to add all the contents from `helperOC` and the `ToolboxLS` to your path. Do this by navigating to the `hw1_code` folder in the explorer window on the left, right-clicking on each folder (e.g., `helperOC`) and choosing "Add To Path → Selected Folders and Subfolders".
- Make sure to also add the HW problem folder to your path too (e.g., `Problem 3` folder should be added to path when running on problem 3 code).
- Check out the `README.txt` inside the zip file for more instructions.

**Problem 1. Deriving the optimal control (30 points).** Consider a robot whose state is position and heading $x := (p_x, p_y, \theta)$ and whose 3D dynamics are:

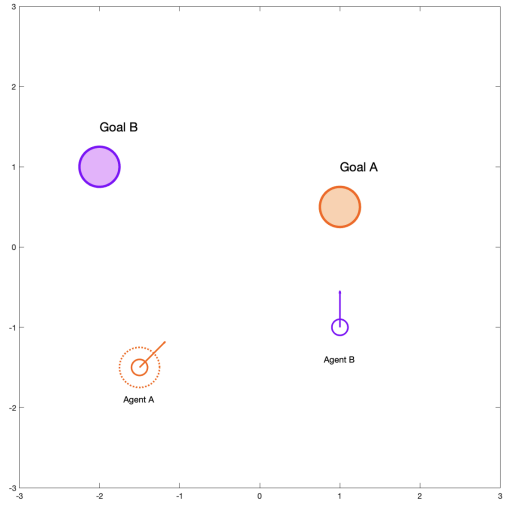
$$\dot{p}_x = v \cdot \cos(\theta) \tag{1}$$
$$\dot{p}_y = v \cdot \sin(\theta) \tag{2}$$
$$\dot{\theta} = \omega. \tag{3}$$

The robot's linear velocity is fixed $v = 1.5 m/s$ and it controls its angular velocity, $u := \omega$ which is bounded, $\omega \in [0.5, 0.5]$. Assume the failure set $\mathcal{F}$ is centered at the origin and contains any state within a 0.5 $m$ radius circle in the $p_x - p_y$ plane irrespective of the heading angle.

1. Mathematically define a signed distance function $\ell(x)$ which encodes $\mathcal{F}$ as the sub-zero level set.
2. Assume the robot is trying to *avoid* the failure set $\mathcal{F}$; derive the optimal control law to steer away from failure. Assume the robot is trying to *reach* the failure set; derive the optimal control law to steer towards failure.




(a) **Problem 3.** Agent A attempts to navigate to goal.




(a) **Problem 3.** Agent B also navigates to their goal.

Figure 1: Environment setup for Problem 3.

**Problem 2. Numerically computing reachable tubes (20 points).** With your favorite numerical toolbox, compute the BRT defined **Problem 1**. In the MATLAB code, fill out `main.m` and `OptCtrl.m` in the `@DubinsCar` class. Visualize the BRT at $\theta = 0, \frac{\pi}{4}, \frac{\pi}{2}$ initial conditions. Explain why the shape of the BRT changes as a function of the initial heading. Submit your code and the BRT plots.

**Problem 3. Multi-agent reachability analysis & safety filtering (50 points).** Consider the pursuit-evasion game between two agents: agent A wants to avoid agent B and agent B seeks to collide with agent A (see left Figure 1). Let agent A's state be their planar position and heading $x^A := (p_x^A, p_y^A, \theta^A)$ and similarly for agent B, $x^B := (p_x^B, p_y^B, \theta^B)$. Each agent's 3D dynamics are:

$$\dot{p}_x^A = v^A \cdot \cos(\theta^A) \qquad \dot{p}_x^B = v^B \cdot \cos(\theta^B) \tag{4}$$

$$\dot{p}_y^A = v^A \cdot \sin(\theta^A) \qquad \dot{p}_y^B = v^B \cdot \sin(\theta^B) \tag{5}$$

$$\dot{\theta}^A = \omega^A \qquad \dot{\theta}^B = \omega^B. \tag{6}$$

Here, agent A's linear velocity is fixed $v^A = 1.5m/s$ and its control is angular velocity, $u^A := \omega^A$. Similarly, agent B's linear velocity is fixed at $v^B = 1m/s$ and they control angular velocity $u^B := \omega^B$. Assume each agent's control actions are bounded by: $\omega^A \in [-0.5, 0.5]$ $rad/s$, $\omega^B \in [-1, 1]$ $rad/s$. Intuitively, agent A moves faster in position space but turns slower; agent B moves slower in position space but turns faster.

In theory, we could solve a reachability problem for the joint 6D system by stacking $x := (x^A, x^B)$ and stacking the dynamics equations above. However, 6D is pushing the limits of what exact grid-based solvers can handle tractably. Instead, we will solve the reachability problem in the *relative state space*, which will reduce the dimensionality to only 3D! Intuitively, think of the relative state space as "how the interaction looks like from the perspective of agent A". Mathematically, let the relative state be $x^{\text{rel}} := (p_x^{\text{rel}}, p_y^{\text{rel}}, \theta^{\text{rel}})$ and the corresponding relative dynamics be:

$$\dot{p}_x^{\text{rel}} = -v^A + v^B \cdot \cos(\theta^{\text{rel}}) + \omega^A \cdot p_y^{\text{rel}} \tag{7}$$

$$\dot{p}_y^{\text{rel}} = v^B \cdot \sin(\theta^{\text{rel}}) - \omega^A \cdot p_x^{\text{rel}} \tag{8}$$

$$\dot{\theta}^{\text{rel}} = \omega^B - \omega^A, \tag{9}$$

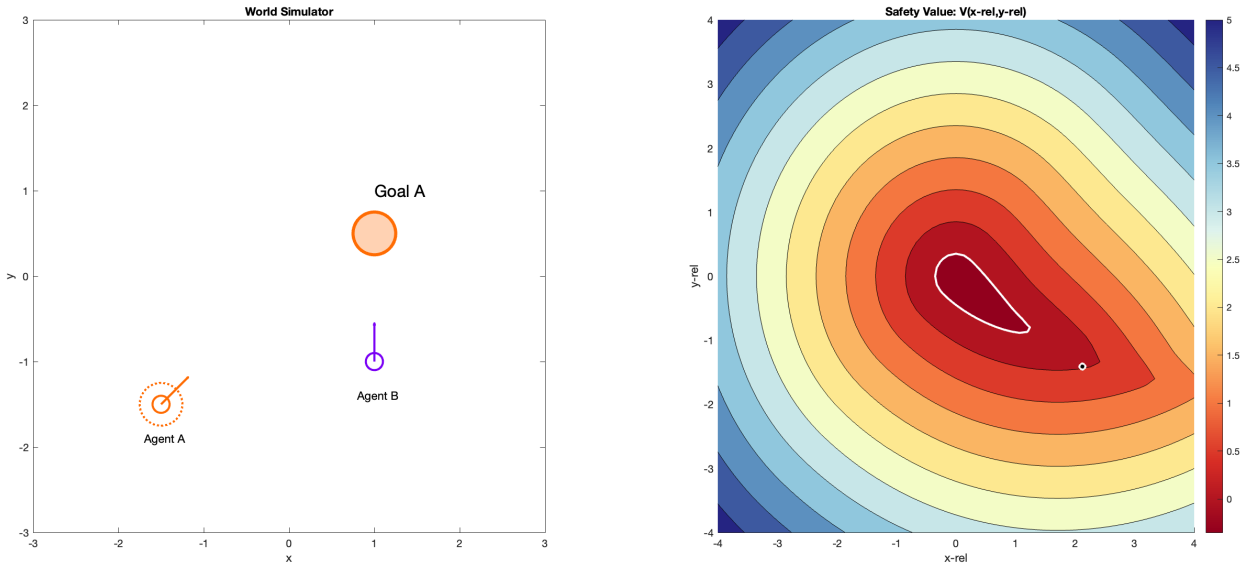



Figure 2: **Hint:** If you computed your value function correctly, the value function visualization at the start of the simulation should look like this.

where just like before, the control action for each agent is $u^A := \omega^A$ and $u^B := \omega^B$.

1. Let the failure set $\mathcal{F}$ be defined by a radius of $0.35m$ starting at the relative x-y origin and being irrespective of the relative angle. Code up this failure set and run the code to compute a BRT by completing the `BRT_computation()` function.

Now, suppose agent A is trying to navigate to a goal area defined by

$$G^A = \{(p_x^A, p_y^A) : \|(p_x^A, p_y^A) - (1, 0.5)\|_2 <= 0.25\},$$

i.e., the goal area is the circle of radius 0.25 $m$ around $(1, 0.5)$ (left, Figure 1)). However, it needs to avoid colliding with agent B. One way to do this is use a nominal planner to steer agent A to their goal without explicitly accounting for agent B's future behavior, and then apply a safety filter whenever required.

2. Using the provided code, first run the provided planner and simulate agent A's trajectory. This code simulates agent B as *optimally adversarial*. Plot the agent trajectories and agent A's control input. Does agent A satisfy the safety constraints?

3. Use the computed BRT from above for safety filtering of the provided planner. Implement a least restrictive controller for the safety filtering by filling out the `get_safety_controller()` function. Plot the agent trajectories and agent A's control input under this new controller. Does the vehicle satisfy the safety constraints?

4. Suppose that agent B is not adversarial; instead, agent B is using the same kind of planner as agent A, but agent B is heading towards goal $(-2, 1)$ with radius 0.25 $m$ (right, Figure 1). During this interaction, simulate agent A as using its original safety filter during interaction. Plot the agent trajectories and agent A's control input. How is agent A's performance impacted in this new interaction?