Documentation - draft

Pietro Pierpaoli

October 13, 2016

Intro

The code provided in the folder is designed to execute both simulations and experiments for the *collision filtering problem*. This guide provide a general reference and introduction to the code, see also comments on MATLAB files for more details.

The code addresses a self-localization problem for a group of N robots located in a partitioned domain. Each partition of the domain is called a *cell*. The object of each robot is to infer what cell it's currently visiting, given the rate of encounter with other robots and the underlying homogeneous dynamics of transitions between cells. The robots are assumed to be circular objects with know radius equal to GGP.

Continuously over time, each robots solve a *go2goal* problem towards a randomly assigned goal. In this way, random trajectories are generated independently by each robots. Every time the robot reaches its goal a new goal is issued. If a Markov transition matrix $M \neq \mathbf{I_m}$ is provided, transitions between cells are also possible. For each cell, the M matrix defines a probability p of staying in the cell and a probability $1 - p$ to move to the following cell. Whenever a transition of cell is drawn, a goal in the new cell overwrites the existing goal.

In order to avoid robots exiting the cells domain while transitioning between cells, an intermediate goal point belonging to the boundary edge between cells is issued. In this way the robots are forced to stay inside the cells' boundaries while transitioning. For this reason, goals are assigned in a two elements queue.

Add notes on collision clusters and the way collisions are resolved.

Description

Run.m Main run file.

buildArena.m The function outputs two objects: a struct ARENA, containing information on the simulation domain, and a list INITCONF, defining what robots occupy each cell at the initial time. For the purpose of simulate real experiments, the code uses two different sets of cells. The first set (whose vertexes are listed in ARENA.GRID) represents the physical domain where the robots move in. These cells are used for plots and together form a connected circular domain. The second set (contained in ARENA.GRID) represents a smaller scaled copy of cells concentric with the first set of cells. Points within the smaller cells are randomly chosen as goal to generate the random trajectory of the robots. In this way, while navigating from goal to goal the robots is prevented from exiting its current cell and experiencing collision with robots in neighboring cells.

dilationPoly Produces a scaled copy of a cell such that a constant gap between the two is respected. The gap between smaller and physical cell is equal to the robot's shape radius times a safety constant.

initHMM Compute the Markov transition matrix for the problem.

Simulation

The simulation create virtual robots named *Kheperas*. The nomenclature comes from the first idea of using Khepera robots for the experiments. Here the name is not meant to refer to the actual Khepera robots.

Khepera Virtual robot object, used to produce the dynamics of the robots. They corresponds to the real moving robots in the experiments and are created in order to maximize the similarity between the code used for simulations and experiments. Initial states are randomly drawn from a uniform distribution over the cell domain and the states are updated accordingly to a unicycle dynamics model.

collisionFinder Produce a list of collision clusters. Given the physical extension of the robotarium agents with respect to available surface, collisions are more likely to happen between group of robots rather then between pairs. Here, this function is used to produce a set of non-overlapping initial positions for the robots. More on this function later.

Experiments

The 'EXPERIMENT' case was build using the robotarium libraries available last year. They might need to be updated.

Main Loop

robotArena Time running loop. Same file for both simulations and experiments. Depending on the number of inputs passed in, simulation or experiments case is detected automatically.

Agent Agent object. Represents the control and estimation software running on the robot. The major function in the object is CONTROLLER, whose general functioning is as follow:

- Every AGENT.CELLDWELL*AGENT.DT seconds, the agent runs the transition Markov process which will either move it to the next cell or not. Each agent also has a state AGENT.TRANSITIONING. Since transitioning from one cell to an other takes some time, the drawing of a new cell is skipped if the agent has not completed the first transition yet.

  A column stochastic matrix has values in each columns that sum up to 1, $\sum_{i=1}^{m} M_{ij} = 1$, for $j = 1, \ldots, m$, where m is the number of cells in the domain. Each entry M_{ij} corresponds to the probability of moving to the cell i given the robot in cell j. Therefore, given the robot is in cell j, the goal is to draw a number from 1 to m such that each of these number have a probability of being drawn equal to $M_{i,j}$. The following technique is used:

  1. The cumulative sum of values for each column produces a vector of increasing real numbers $c = \{M_{1,j}, M_{1,j} + M_{2,j}, \ldots, 1\}$. Value of c is computed using CUMSUM(AGENT.M(:,AGENT.CELL);
  2. A random number z from a uniform distribution on the $(0, 1)$ interval is drawn;
  3. Each entry of c are compared with z. Some entries of c will be greater than z and other smaller. The target cell will be i^{th} cell, where c_i is the first entry of c greater than z.

  If a new cell is drawn two goals are assigned. The first is an intermediate goal (*transition goal*) along the edge boundary of the two cells, while the second is the actual goal in the new cell. In this way agents are forced to stay in the cell domain rather than cut through the middle.

- If the distance to the goal is lower than a tolerance, we first evaluate if the agent was transitioning between cells. If it was, we assume the transition complete. Finally we move the second goal to the first position and add a new goal to the queue.

ArenaPlot Produce and updates the main simulation plot.