Socially Compliant Path Planning

Abhijat Biswas, Ting Che Lin, Sean Wang

I. Introduction

Robots that share space with humans are becoming increasingly common. Mobile robots especially, need to be aware of humans and associated conventions when interacting with them. Hence, humans cannot be treated in the same way as moving obstacles that are to merely be avoided. For example, robots must know not to cut-off moving pedestrians and instead, either yield to them or take an alternate route.

II. Overview

We wanted to use representative pedestrian trajectory data. So, we use the **Stanford Drone Dataset** for real pedestrian data. A sample frame are in Fig. 1.

The dataset has videos recorded from an aerial view with individual pedestrain bounding boxes annotated over time.

A. Math formulation

We formulated our problem as a time-limited horizon search-based planning problem on a grid world. See Fig 1

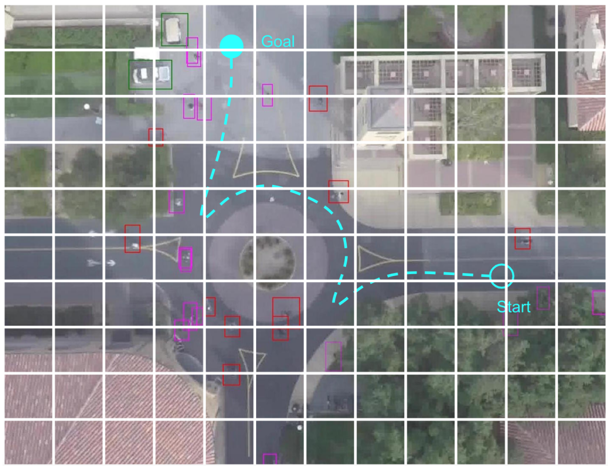


Fig. 1: Example frame and formulation. Gridworld is illustrative. Actual gridworld is at pixel level

More details about the planner are in Section IV.

B. Software

Our planning algorithm is written in C++ and the rest of our system, including the Social-LSTM based trajectory prediction, costmap calculation, and simulation is in Python. We use the brilliant XTensor and pybind11 to allow our C++ planner to modify the python side costmap. This combination allows us to communicate between C++ and Python in a non-copying fashion.

III. Pedestrian Trajectory Forecasting

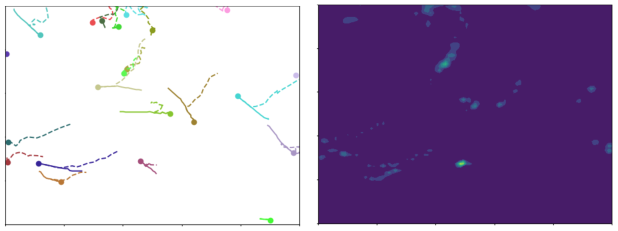
Our trajectory forecasting pipeline utilizes Social-LSTM[1] network for pedestrian path prediction. The network is trained on the deathCircle, gates, coupa, and nexus scenes of the Stanford Drone dataset[2] for 20 epochs. We used the modified version of the network implemented by Anirudh Vemula[3].

The network directly predicts the Bivariate Gaussian distribution of the future location of the pedestrians, or, more simply, the probability that the pedestrian will be at a certain location at a specific time. The future trajectories for every i pedestrian is obtained by sampling the predicted distribution.

$$trajectory_i \sim p(future\ trajectory_i | past\ trajectory_i) \tag{1}$$

For this project, in order to demonstrate real time demo, we pre-computed the predicted distribution of the future trajectories and saved it to disk. During planning, we then queried the predicted distribution at the specified time stamp and utilizes the trajectory distribution as the Social Cost function. The Social Cost at every time steps for N pedestrians can be calculated as the following.

$$c_t = \sum_{i}^{N} p(future\ trajectory_{i,t} | past\ trajectory_{i,t}) \tag{2}$$



(a) (b)

Fig. 2: (a) Sampled Prediction Trajectory (b) The Corresponding Social Cost

IV. Path Planning

A. State and Cost

Since the trajectory prediction of the pedestrians are time dependent, the cost of movement of the robot will not only

[1]http://cvgl.stanford.edu/papers/CVPR16_Social_LSTM.pdf
[2]http://cvgl.stanford.edu/projects/uav_data/
[3]https://github.com/erichhhhho/social-lstm-pytorch

depend on the robot's location, but also the time of the movement. For this reason, the state of the robot needs to include time when planning ($s = (x, y, t)$). For the cost function, we wanted to include the distance traveled as well as the likely hood of running over a pedestrian. To do this, we used the following cost function.

$$c(s_1, s_2) = D(s_1, s_2) + P(x(s_2), y(s_2), t(s_2)) \quad (3)$$

$D(s_1, s_2)$ gives the euclidean distance between states s_1 and s_2. $P(x(s_2), y(s_2), t(s_2))$ gives the social cost of moving to state s_2 and can be calculated using the previously mentioned method.

B. Algorithm Used

For this problem, since we could only predict human paths into the near future, we wanted to use a search algorithm that searched for partial paths into the near future. We decided to use a variation of RTAA*, where a number of states are expanded, then the robot follows the optimal path to the state $s = argmin_{s' \in open} g(s') + h(s')$, or the state in the open set with the lowest f value. However, in our variation, the number of states expanded each time was not fixed, instead the expansion of states keeps happening until either the next state to expand has a time that is greater than the last prediction time, or is the goal. By doing so, the path returned will either bring the robot to the goal, or will be a partial path, with a low cost of traversal, to a state with a low heuristic value. If it returns a partial path, it will take the same amount of time to traverse as time until the last predition.

For the heuristic, we ignored time and the predictions and just used euclidean distance. This is guaranteed to be admissible since the prediction values are non-negative. Furthermore, this heuristic is consistent. Since we know that euclidean distance satisfies the triangle inequality, $h(s) \leq D(s, succ(s)) + h(succ(s))$. Furthermore, since we know that the cost is greater than euclidean distance, $h(s) \leq c(s, succ(s) + h(succ(s))$.

The pseudo-code for the search algorithm used is in Algorithm 1

Algorithm 1 Partial Path Search Algorithm

```
1:  procedure SEARCH($s_{start}, s_{goal}, P$)
2:      $t_{limit}$ = time of last prediction
3:      $s = argmin_{s' \in open} g(s') + h(s')$
4:      while $s \neq s_{goal}$ and $t(s) < t_{limit}$ do
5:          remove $s$ from open and insert into closed;
6:          for every $s' \in succ(s)$ that is not in closed do
7:              if $g(s') > g(s) + c(s, s')$ then
8:                  $g(s') > g(s) + c(s, s')$;
9:                  insert $s'$ into open
10:         $s = argmin_{s' \in open} g(s') + h(s')$
11:     backtrack from $s$ and return path to robot
```

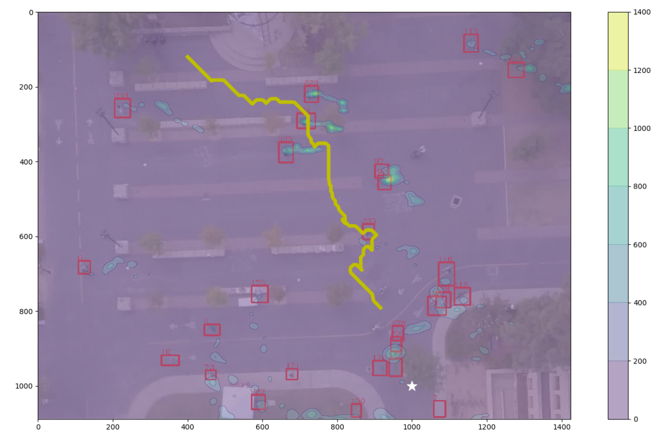


Fig. 3: Intermediate path plan with overlaid pedestrian trajectory prediction-based costmaps. The goal is at the white star (bottom right)

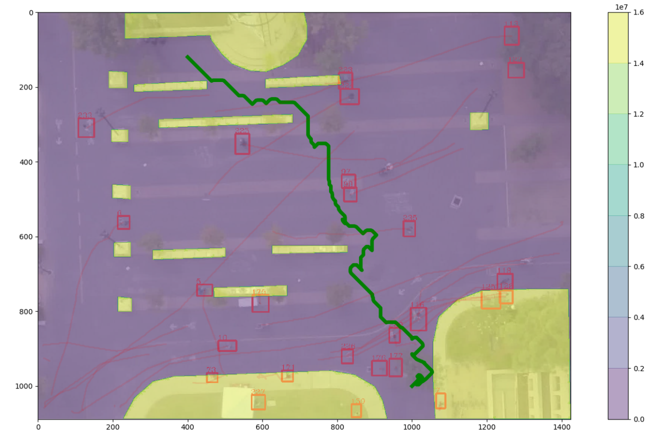


Fig. 4: Sample completed path plan. Faint red lines show the

V. Results and Discussion

Average planning times: Typical planning times are under 50 $milliseconds$. However, in the worst case (densely packed locations with lots of obstacles and multiple trajectory predictions) we have observed worst case planning times upto 6 $seconds$. We plan for a horizon of 4 $seconds$. The average planning time is 0.51 $seconds$

Our **grid sizes** are the same as the pixel resolution of the frames from the camera. For the scenario in Fig 4, this is 1088×1424.

VI. Demo and Code

Demo video is here. Code can be found here, including a README with instructions to run it.