April 10, 2014

# Robot Navigation in Dense Human Crowds
Final Report

Neil Traft
University of British Columbia

## Overview

In their 2010 IROS publication [1], Trautman and Krause develop a path planning algorithm that is safe and yet does not suffer from the "freezing robot problem" (FRP). Their method consists of a model of crowd interaction combined with a particle-based inference method to predict where the crowd (and the robot) should be at some time $t+1$ in the future. The idea is that if one can develop a reliable model of intelligent agents in a crowd, and include the robot as just another of those intelligent agents, then the predictions of the model yield the robot's future path.

The goal of this project is to reproduce their results in simulation on the original dataset. Given some annotated video of pedestrians in a crowd, we can choose one of the pedestrians to represent the robot, and compare the path planned by the robot to the actual path taken by the pedestrian.

### Interacting Gaussian Processes

The crowd interaction model is a nonparametric stochastic model based on Gaussian processes, dubbed *Interacting Gaussian Processes* (IGP). In IGP, the actions of all agents, including the robot, are modeled as a joint distribution:

$$p(\mathbf{f}^{(R)}, \mathbf{f} | \mathbf{z}_{1:t})$$

where $\mathbf{f}^{(R)}$ is the robot's trajectory over $T$ timesteps, $\mathbf{f}$ is the set of all human trajectories, and $\mathbf{z}_{1:t}$ is the set of all observations up to the current time point. For the purposes of this algorithm, observations of human and robot position are taken to be more or less perfect, since we are only trying to solve the navigation problem, not situational awareness.

At each timestep, each agent's new position is represented as a random variable from a probability distribution. It is important to note that this distribution is *not* Gaussian, due to two major additions to avoid the uncertainty explosion which leads to the FRP (see Figure 1).

First, goal information is given as a final "observation" at time $T$, resulting in the full set of observations $\mathbf{z}_{1:t,T}$. The robot's goal, $y_T^{(R)}$, is known and can be added with good confidence. The goals of other agents can be omitted or can be added with a high variance, to encode how uncertain we are about the goal.

The second addition IGP makes to standard Gaussian processes is the inclusion of an "interaction potential",

$$\psi(\mathbf{f}^{(R)}, \mathbf{f}) = \prod_{i=1}^{n} \prod_{j=i+1}^{n} \prod_{\tau=t}^{T} \left(1 - \alpha \exp\left(-\frac{1}{2h^2} |\mathbf{f}_\tau^{(i)} - \mathbf{f}_\tau^{(j)}|\right)\right)$$

In essence, this potential grows very small whenever two agents $i$ and $j$ become very close at any time $\tau$. This has the result that any set of paths where agents become too close is treated as very unlikely. The parameter $h$ controls the desired "safety distance" and $\alpha \in [0, 1]$ controls the "repelling force". Thus, the final posterior is given as:

$$p_{IGP}(\mathbf{f}^{(R)}, \mathbf{f} | \mathbf{z}_{1:t}) = \frac{1}{Z} \psi(\mathbf{f}^{(R)}, \mathbf{f}) \prod_{i=1}^{n} p(\mathbf{f}^{(i)} | \mathbf{z}_{1:t})$$

The above is a nonlinear, multimodal distribution, so it can't be sampled directly. Instead, we sample from Gaussian priors $p(\mathbf{f}^{(i)} | \mathbf{z}_{1:t})$ and resample weighted by our desired distribution (a particle filter). This is described in the next section.

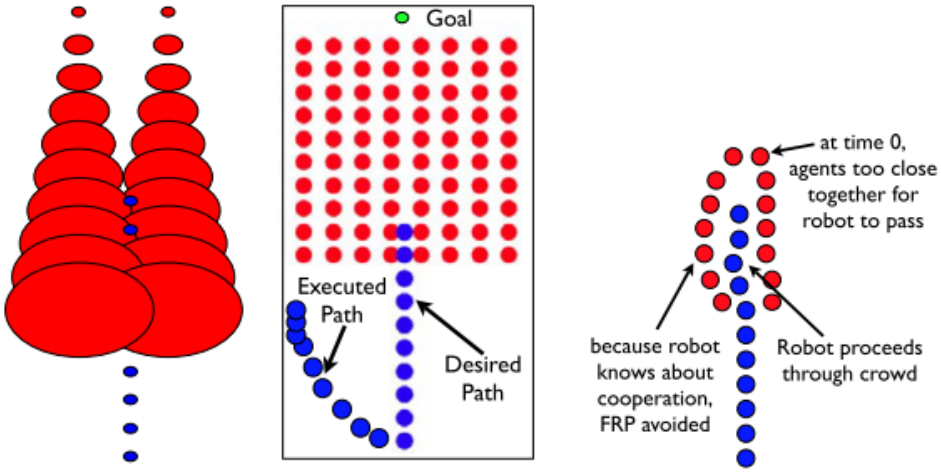



**Figure 1** Diagrams taken from [1] describing the uncertainty explosion which leads to the Freezing Robot Problem. **Left:** Depicts the uncertainty explosion in standard motion models, where each agent's trajectory is *independent* from the others. **Middle:** A demonstration of why even *perfect* prediction, devoid of uncertainty, can still lead to the FRP. In crowded environments, *all* paths can have a high cost function, leading to extreme evasive maneuvers or freezing. **Right:** The ideal model, based on the insight that intelligent agents engage in *cooperative* collision avoidance.

### Importance Sampling

Now that we have a model, we wish to sample from it and take the mean as the desired path. Since we can't sample from it directly, we instead use the *importance sampling* technique which is widely used in particle filters. Each sample is weighted by the ratio of the IGP to the basic GP (i.e. the Gaussian distribution, without the interaction potential):

$$\begin{aligned} w_i &= \frac{p_{IGP}}{p_{GP}} = \frac{p_{IGP}((\mathbf{f}^{(R)}, \mathbf{f})|\mathbf{z}_{1:t})}{\prod_{j=R}^{n} p((\mathbf{f}^{(j)})_i|\mathbf{z}_{1:t})} \\ &= \frac{\psi((\mathbf{f}^{(R)}, \mathbf{f})_i) \prod_{j=R}^{n} p((\mathbf{f}^{(j)})_i|\mathbf{z}_{1:t})}{\prod_{j=R}^{n} p((\mathbf{f}^{(j)})_i|\mathbf{z}_{1:t})} \\ &= \psi((\mathbf{f}^{(j)})_i) \end{aligned}$$

where $(\mathbf{f}^{(j)})_i$ is a single sample from the trajectory of agent $j$.

Given this formulation for $p_{IGP}$ and an appropriate weighting $w_i$ for each sample, the ideal paths can now be expressed as the *maximum a-posteriori* (MAP) assignment for the posterior,

$$(\mathbf{f}^{(R)}, \mathbf{f})^* = \arg\max_{\mathbf{f}^{(R)}, \mathbf{f}} \left( \sum_{i=1}^{N} w_i (\mathbf{f}^{(R)}, \mathbf{f})_i \right)$$

where $(\mathbf{f}^{(R)}, \mathbf{f})_i$ is a set of samples from the Gaussian processes $(\mathbf{f}^{(j)})_i \sim \mathrm{GP}(\mathbf{f}^{(j)}, m_t^{(j)}, k_t^{(j)})$. The total number of samples is $N$ and we take the robot's next position to be $(\mathbf{f}_{t+1}^{(R)})^*$. To approximate the optimal robot path $(\mathbf{f}^{(R)})^*$, we take the mean path over all samples after importance resampling:

$$(\mathrm{f}_t^{(R)})^* = \frac{1}{N} \sum_{i=1}^{N} (\mathrm{f}_t^{(R)})_i$$

## Implementation

The project is implemented in Python, using the OpenCV library [2] for visualization and video playback (but not for any of the vision algorithms it provides).

### The Dataset

The dataset to be used is the ETH Walking Pedestrians (EWAP) dataset from [3]. It can be obtained from [4].

The dataset contains two annotated videos of pedestrian interaction captured from a bird's eye view. The one used depicts pedestrians entering and exiting a building.

The main annotations are a matrix where each row gives the position of an agent $i$ in a particular video frame $t$. Thus for each frame $t$ we have potentially multiple pedestrian observations $\text{pos}_t^{(i)}$, and this forms our observation at time $t$:

$$\mathbf{z}_t = \text{pos}_t^{(1:n)}$$

where one of the $n$ pedestrians is chosen to represent the robot $R$. The velocities are not used in the present IGP formulation.

The positions and velocities are in meters and were obtained with a homography matrix $H$, which is also provided with the annotations. To transform the positions back to image coordinates, it is necessary to apply the inverse homography transform:

$$^m\text{pos}_t^{(i)} = H_{mw}^{-1} \cdot {}^w\text{pos}_t^{(i)}$$

$$m = \text{image}, w = \text{world}$$

The intrinsic camera parameters are not needed; it is presumed that they are included in the camera matrix provided by the dataset. There is also no translation needed; the origin of the world can be taken to be the (transformed) origin of the image without loss of generality. Thus, pixel coordinates can be expressed as

$$r = -f_x \frac{x}{z} = \frac{u}{z} \qquad c = -f_y \frac{y}{z} = \frac{v}{z}$$

where $f_x$, $f_y$ are the intrinsic camera parameters and $x$, $y$ are coordinates in the image frame. So to obtain pixel coordinates from image coordinates it is necessary only to normalize so that $z = 1$:

$$\begin{pmatrix} r \\ c \\ 1 \end{pmatrix} = \begin{pmatrix} {}^m x/z \\ {}^m y/z \\ {}^m z/z \end{pmatrix}$$

## Technical Hurdles

There are some nuances to using a Gaussian process that can stun and bewilder those who are new to them. IGP is very sensitive to these nuances, and an understanding of Gaussian processes is essential to getting it to work.

In addition to this, there are other parameters to be tuned and implementation choices which are left open-ended by the original author. These choices comprise the hurdles one has to clear to see good results from IGP.

### The Covariance Function

The covariance function (also known as a *kernel*) used for the Gaussian processes in IGP is an absolutely crucial consideration. However, the authors of [1] do not describe their choice of kernel. In their 2013 follow-up work, they mention the class of kernel used, but do not go into sufficient detail to reproduce their implementation. [5]

What is needed from our Gaussian process is a prior which is straight most of the time, but with some curviness. This encodes our assumptions about the way humans move. People almost always walk in straight lines, so we need a Gaussian process that gives us that. We also would like some curviness for those times when people don't walk straight, and we also need to encode the noisiness of our observations. [1]

From these requirements, we can see that the traditional kernel for Gaussian processes won't be suitable. The general-purpose kernel is the squared exponential (SE) kernel,

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\Big( -\frac{1}{2\ell^2} |\mathbf{x} - \mathbf{x}'|^2 \Big)$$

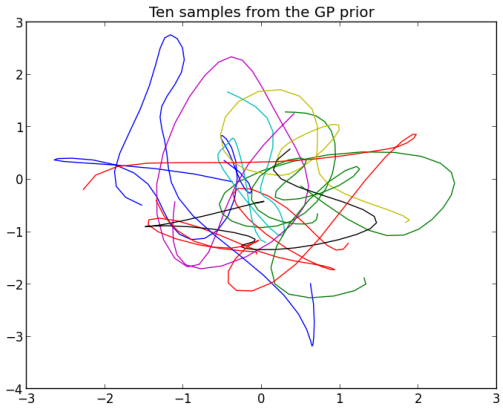
where $\mathbf{x}$ and $\mathbf{x}'$ denote two vectors of independent variables whose covariance we wish to calculate, and where $\sigma^2$ and $\ell$ are the noise and length hyperparameters, respectively.

First of all, the SE kernel is too variable; too "curvy". An example is given in Figure 2. The aimlessly meandering paths seen there do not resemble a person enroute to a destination. Moreover, a much bigger problem is the fact that SE is *stationary* (shift invariant). Informally, this means that it will gravitate toward the mean wherever there is no data. Observe how it hovers about the origin in the example.

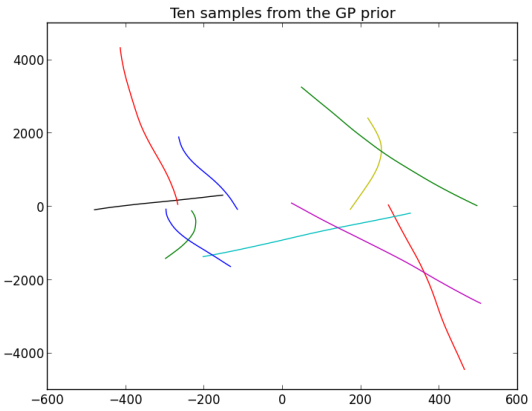
In contrast, the linear regression kernel is a *non*-shift-invariant kernel which satisfies our need for a straight path to continue in a straight line in the absence of data. Also known as a dot product covariance function, it is expressed as

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 + \mathbf{x} \cdot \mathbf{x}'$$

[1] From correspondence with the author, Peter Trautman.




**(a)** A summed kernel composed of squared exponential and noise kernels.




**(b)** A summed kernel composed of Matérn, linear regression, and noise kernels.

**Figure 2** The importance of choosing a proper covariance function. (4a) This is much too curvy and loopy to represent a human path. Humans generally have destinations; they don't wander aimlessly. (4b) We consider this to be a much better model of human movement. Human paths are mostly straight, but with occasional curviness.

Having only straight lines generalizes to simple linear regression, however, which is obviously not desirable. We still need to be able to approximate nonlinear paths, so we add in one of the Matérn class of covariance functions defined in [6],

$$k(r) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right)$$

where $r = |\mathbf{x} - \mathbf{x}'|^2$. This gives us nicely shaped paths, and we finally add in a noise kernel, which is just a constant variance applied to our training data. The final choice of kernel is illustrated in Figure 2, and an example of sampling from the posterior can be seen in Figure 3.

### Kernel Hyperparameters

Perhaps equally crucial to the choice of covariance function is the choice of *parameters* for the covariance function. These include such parameters as the variance $\sigma^2$ and the length-scale $\ell$. The variables are often co-dependent [2] and highly dependent on the data you wish to approximate. Generally, they should be learned from the data. [6]

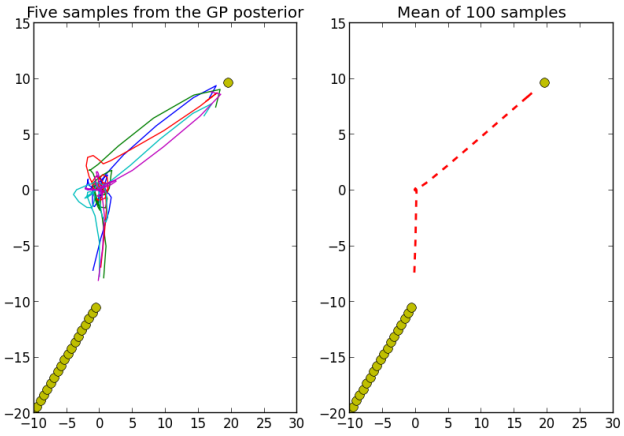
I was able to obtain the original hyperparameters used for the EWAP dataset from Peter Trautman, so I did not have to learn the hyperparameters myself. Even so, producing good paths was not automatic.

Hyperparameters are directly dependent on the scale of coordinate system. This means that the hyperparameters must be tuned to the units that the data is given in. Hyperparameters learned in meters cannot be used in millimeters or image pixels: the scale of the parameters must match the scale of the data. Originally I implemented the algorithm in the world coordinate frame, and I had to switch to the image coordinate frame to match the original implementation. Figure 3 shows an exploration of the effect of hyperparameters on predicting a simple, straight-line path.

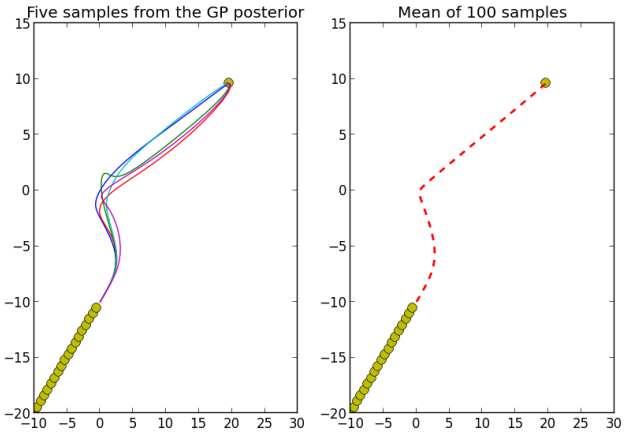
### MAP Approximation

As mentioned in the Importance Sampling section above, the next waypoint of the robot is given by the MAP assignment of the posterior distribution. However, it is not clear how this is obtained in the original work. The answer is that we take the mean of the resampled paths in order to approximate the MAP assignment. This is roughly equivalent to taking the weighted mean of the Gaussian process samples, prior to resampling.

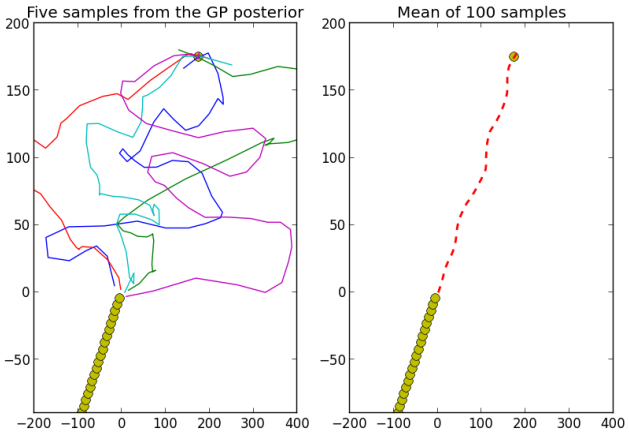
[2] Meaning that if you fix one variable, the optimal choice for the other variable depends on the value of the fixed variable.



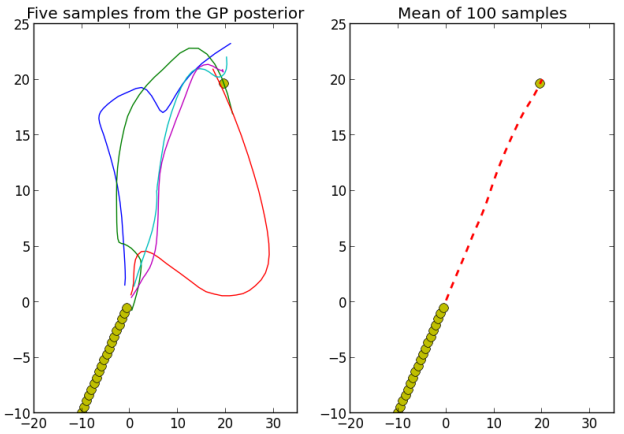

**(a)** Poor choice of covariance function (squared exponential) and poorly tuned hyperparameters. Notice the attraction toward the mean, $(0, 0)$.




**(b)** Improved hyperparameters are still not enough to overcome a fundamentally incorrect covariance function.




**(c)** Good choice of covariance function (Matérn + linear + noise) brings us much closer to the mark, but badly tuned hyperparameters still makes things difficult.




**(d)** Good covariance function combined with good hyperparameters.

**Figure 3** The importance of choosing proper hyperparameters. Hyperparameters must be learned to properly match what we expect from human and robot movement patterns. Hyperparameters must be further tuned for a particular scene, since they are closely related to the scale of our coordinate frame.

Practically speaking, this is a shortcut. The reason for it is that the path with the largest weight is still unstable unless we take a very large number of samples. [3] However, this potentially destroys multimodal distributions, which is why we used a particle filter in the first place. This seems to work well enough in this example, but an idea for improvement is described in Next Steps.

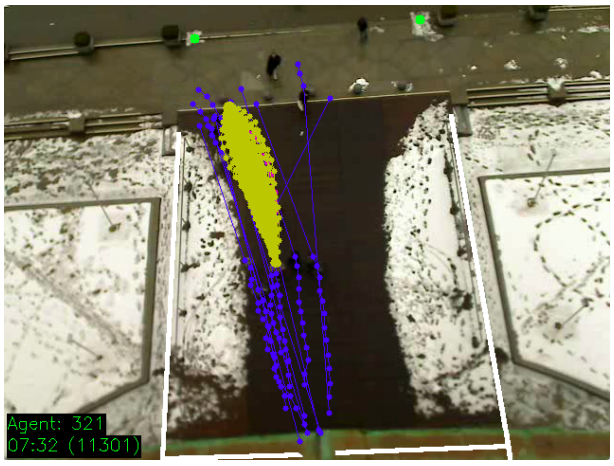
[3] This sentiment was conveyed by Peter Trautman via personal correspondence.

## Experimental Results

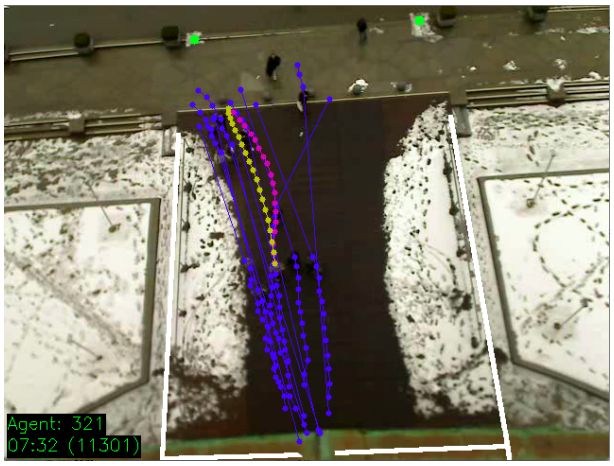
I evaluated IGP on each of twelve pedestrians, in turn, taken from a small segment of video which exhibited high crowd density and interaction. See Figure 4 for a screenshot of the scene.

The dataset and how it is annotated is already outlined above. Once I had the ability to process this data I set up an automated experiment. For each pedestrian, IGP simulated a full execution of, *not* the pedestrian's own path, but

the path planned by IGP. The simulated "robot" started and ended at the same location as the pedestrian, but at all intermediate timepoints the robot replanned the path as if it were at the location planned previously. Timesteps were fixed so that the pedestrian and the robot both took the same amount of time to reach the end goal (so it was assumed that the robot could move as fast as the human).




**(a)** The samples from the posterior distribution (after resampling according to the interaction potential).




**(b)** The path ultimately planned (yellow), compared with the pedestrian's own path (pink).

**Figure 4** Sample output of the final product. Blue paths represent each agent's path up to the current time point, as well as their ultimate destination.

At the end of each run, the robot's performance is compared to that of the human's in terms of two metrics: *path length* and *path safety*. My results are summarized in Figure 5 and the results of [1] are shown in Figure 6. In [1], IGP outperformed the human pedestrian in both metrics, while in my own experiment, it only outperformed in terms of path length, and not safety. However, the robot never was closer than the minimum observed distance of two pedestrians over all runs (about 10 pixels). Thus, we conclude that the paths planned by IGP were safe and comparable to those of the human.

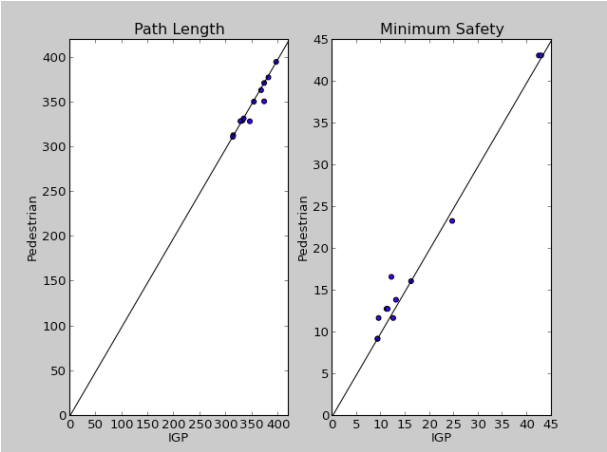



**Figure 5** The results of the algorithm performed on 12 pedestrians from a particularly crowded segment of the video. Observe that IGP almost always finds an equal length or shorter path than the pedestrian. However, the pedestrian usually maintains a larger distance from other agents, though IGP never goes below the lowest observed actual separation (~10 pixels), so it is considered safe.

The experiment was run with the minimum number of particles needed for good results (anecdotally, 100 particles). This took an average of 2.0 seconds to plan a single path on a laptop. The original paper does not specify the number of particles used in their experiment, so this is quite possibly the reason for the difference in performance. There are also a number of shortcomings which may have additionally contributed to the degraded performance. These are outlined in the next section.

## Next Steps

First and foremost, I believe there are still bugs in my implementation of Gaussian pro-

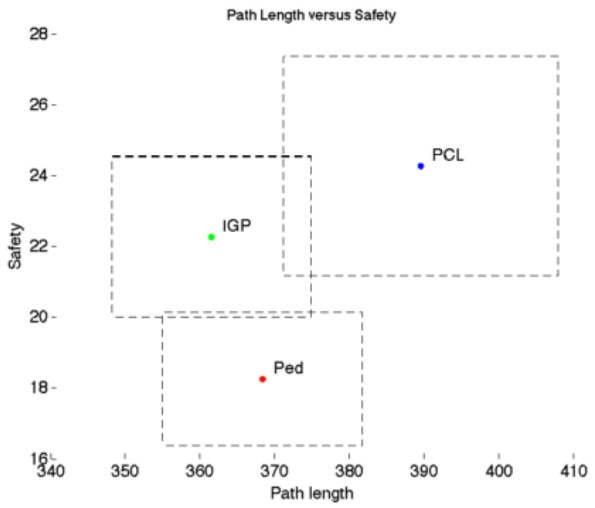



**Figure 6** The results reported in the original paper, compiled from 10 trials. Here, we see that IGP outperforms human pedestrians in both path length and path safety. We aren't told which 10 pedestrians are evaluated in the original paper, so these results are somewhat anecdotal and will not necessarily align with our own.

cesses that need to be cleaned up. It generally works, but I have not been able to confirm that the covariance functions are completely correct. Peter Trautman has expressed that the paths do not look quite right. It is nice to know that IGP performs well despite not having perfect Gaussian process priors. It does seem like it has degraded performance, however, and should be fixed.

It is also possible that the performance has not been fully evaluated, or that the results reported in [1] are optimistic. In the near future, I would like to run more experiments to fully evaluate the algorithm's performance over a number of different crowd densities. There was only time to run the single twelve-trial experiment for this project.

Another major avenue I would like to explore is how to learn the hyperparameters for the Gaussian process. I do not think this is too difficult, but I did not have time to do it for this project. The process is outlined in Chapter 5 of [6] and Peter Trautman has provided me with an example script. Knowing how to do this would be necessary before trying the algorithm in any other environment other than this exact data set.

Along the lines of learning hyperparameters, my current implementation uses the same set of parameters for both human and robot agents. But humans and robots do not move in the same way (robots are generally less agile). So it follows that the best algorithm will learn separate parameter sets for human and robot.

There are a number of other free variables in IGP, other than those of the covariance functions. The interaction potential has two in particular that may have a significant impact on the performance of the path planner. The distance into the future to plan a path is also an open avenue of research. The original IGP predicts a fixed 30 timesteps into the future, while mine predicts the exact number of timesteps needed to reach the goal (since my implementation runs on annotated video, it can cheat by knowing the future). An interesting avenue for research would be to create a universal IGP algorithm, where all these free variables, hyperparameters included, could be learned online.

Another open problem is that of assigning destination waypoints to the other agents in the scene. This step is fairly important for controlling the variance on agents' paths and getting sensible results from IGP. The better we can guess where someone is headed, the better we can predict how they're going to get there, and this could be a good research question to answer.

Finally, I would like to speed up the implementation. The algorithm presented here took about 2.0 seconds on average to plan a single path, with 100 particles. The original paper cites times of 0.1 seconds for the same number of particles, so there is much room for improvement here. The original paper was written in Matlab but I do not believe the language of implementation is the bottleneck. More likely, the GPML toolbox's implementation of Gaussian processes is much more efficient than my own (some parts of the toolbox are optimized in C).

## Conclusion

In this project I have demonstrated the feasibility of the Interacting Gaussian Processes algorithm for path planning through human crowds.

From these preliminary experiments, it seems that IGP has potential as a tractable method for searching through an infinitely large state space of actions.

However, there is more work to be done. I was not able to match the author's reported performance, neither in path planning nor in computational efficiency. I have noted some immediate improvements that could be made to my implementation which would likely close this gap. Still, I've also demonstrated IGP's fragility in terms of its heavy reliance on parameter tuning. A compelling research problem for the future would be to create a version of IGP which learns these parameters online.

## References

[1] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 797–803, Oct. 2010.

[2] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[3] S. Pellegrini, a. Ess, K. Schindler, and L. van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," *2009 IEEE 12th International Conference on Computer Vision*, pp. 261–268, Sept. 2009.

[4] "Ethz - computer vision lab: Datasets." http://www.vision.ee.ethz.ch/datasets/index.en.html. Accessed: 2014-03-01.

[5] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: the case for cooperation," *2013 IEEE International Conference on Robotics and Automation*, pp. 2153–2160, May 2013.

[6] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.