RGDB Enabled Human Centric Navigation

Liz Murphy and Peter Corke
CyPhy Lab, Queensland University of Technology
Brisbane, Australia
email: liz.murphy@qut.edu.au

Abstract—This work details the use of RGBD cameras in creating a human centric navigation system. This system leverages the OpenNI framework's skeleton tracker to build up a representation of a robot’s operating environment that is learned purely from the actions of humans in that environment. An RGBD sensor mounted on the robot performs the detection and tracking of people moving through the environment. Trajectories learned as the robot follows people are then clustered online and organised into a tree-like probabilistic data structure which can be used to detect anomalous trajectories. We then reverse-engineer a costmap from the clustered trajectories and use this costmap to inform the robot’s on-board planning process. Results show that the resultant paths taken by the robot mimic expected human behaviour and can allow the robot to respond to altered human motion behaviours in the environment.

I. INTRODUCTION

A robot navigating in an environment densely populated by people can benefit from an awareness of human behaviour that is not easily gleaned from standard sensors. It is easy to envisage situations in which people learn by observing those around them and alter their behaviour accordingly; if something is spilt on the floor, the crowd will divert around it, or if the flow of human traffic is down the left hand side of a corridor it would be a foolish person who chooses a path down the right. Incorporating this sort of information into navigational maps used by robots is near impossible when relying on sensors like vision and laser alone; instead what is most often used is an occupancy grid representation where all open space is viewed as equally good. In this work we propose an online method of creating human-centric navigational maps by following people through the environment.

This paper makes 3 key contributions to the problem of learning human preferred paths through the environment. The first is a People Tracker ROS package that leverages the OpenNI [8] library's open source Skeleton Tracker to allow a mobile base to safely and persistently follow a person at normal walking pace. The second is an online clustering technique that builds a skeletal representation of the trajectories taken by people through an area. Its probabilistic foundation means that it requires limited tuning, and can incorporate individuals different walking speeds easily. The final contribution is a map creation tool that takes the trajectory clusters and builds a navigational map that is heavily biased towards locations that people walk on in an environment. Coupled with the clusterer, we have a fast online map generation process that can quickly adapt to changed behaviours of people in the environment. What differentiates our approach most from existing approaches is the online learning aspect, as well as the trajectory gathering using a moving platform and RGBD camera.

II. METHOD

A. People Following

The first step is to obtain a set of trajectories by detecting people walking through the environment and then following them to obtain a trajectory. To detect and track people, we use the API provided by OpenNI [2] to interface to the User Generator middleware that generates a representation of a body in the 3D scene. This allows us to pick out the location of a given user’s torso on a frame-to-frame basis.

We then use a pure pursuit approach [1] to follow the person. It operates by calculating an error term

\[
e = \sqrt{(x^* - x)^2 + (y^* - y)^2 - d^*}
\]

which is the difference between the desired following distance \(d^*\) and the current distance of the robot from the person at offset \((x, y)\) in the robot frame.

From this, we use a basic Proportional-Integral controller with gain terms \(K_i, K_e\) to set the robot’s desired forward velocity

\[
v^* = K_e e + K_i \int e dt
\]
The bearing of the person relative to the robot is
\[
\theta^* = \tan^{-1} \frac{y^* - y}{x^* - x}
\]  
(3)
and the difference between that and the robot’s current heading \( \theta \) is used to set the angular velocity
\[
\alpha = K_h (\theta^* \ominus \theta)
\]  
(4)
with a proportional controller gain \( K_h > 0 \) and where \( \ominus \) denotes the smallest difference on \( S \).

B. Trajectory Clustering

The trajectory clustering algorithm is based on [3], but modified to deal with the trajectories being sourced from a mobile platform rather than from fixed down-looking overhead cameras as in the original paper.

Central to the algorithm is the notion of raw trajectories, which embody the instantaneous locations \((t^*_i, t^*_y)\) of the person being followed at time \( i \), and clusters which aggregate together similar trajectories in a probabilistic representation \((c^*_x, c^*_y, c^*_{\sigma_x}, c^*_{\sigma_y})\) at time \( j \). We assume trajectories are continuous in time.

The algorithm has two parts: tree building and a tree maintenance phase. The former is depicted as a state machine in Figure 2.

- **A New Trajectory** is considered to appear on start up, or when a significant discontinuity appears in the input to the clusterer. We allow a new trajectory \( T \) to reach a minimum size \( l_{new} \), and then compare it to existing branches \( C \) in the cluster tree using a distance measure

\[
D(T, C) = \frac{1}{n} \sum_{i=1}^{n} d(t_i, C)
\]  
(5)
where

\[
d(t_i, C) = \frac{\text{dist}(t_i, \hat{c})}{\sigma_j^2}
\]  
(6)

\( \hat{c} = c_j \) s.t. \( j = \arg \min_{j=1}^{n} \text{dist}(t_i, C_j) \)

and \( \text{dist}(t_i, c_j) \) is the Euclidean distance from point \( t_i \) on the trajectory to a point \( c_j \) on the cluster. If the distance between the new trajectory and the closest existing cluster is found to be less than some threshold level \( d_{siib-Thresh} \) then we begin updating the matching cluster. Otherwise, we start creating a new cluster.

- **In the Creating state**, points from the person’s trajectory are continually added to a new, temporary cluster. The distance between the last point added to the cluster and the penultimate point is continually monitored, and if it exceeds a threshold level \( \text{StepPrThresh} \) we assume a new trajectory has begun. New clusters are thus added to the cluster tree in a *delayed* fashion. Once we have a complete trajectory we prune it in a probabilistic fashion, to ensure a minimal representation of the trajectory is used.

- **While Updating** an existing cluster the incoming trajectory is used to update, in a weighted average, the closest point \( \hat{c} = (\hat{c}_x, \hat{c}_y) \) on the existing cluster

\[
\hat{c}_x = (1 - \alpha)\hat{c}_x + \alpha t^*_x \\
\hat{c}_y = (1 - \alpha)\hat{c}_y + \alpha t^*_y \\
\hat{c}_{\sigma_x} = (1 - \alpha)\hat{c}_{\sigma_x} + \alpha (\text{dist}(t_i, \hat{c}))^2
\]  
(7)
The parameter \( \alpha \in [0, 1] \) allows the rate at which clusters fit to newly detected data to be moderated. The trajectory-cluster distance of Equation 5 is continually monitored. If it exceeds a threshold level \( \text{Drift}_{thresh}, \) or the end of the trajectory is reached, we clear the trajectory and move to the **Splitting** state.

- **In Splitting** we check to see if there are any child nodes of the previously-matched cluster and transition to either **Creating** or **Updating** depending on the result.

While tree building operates on a frame-by-frame basis, tree maintenance occurs only periodically. It involves 3 operations:

- **Merging** traverses levels of the tree and uses a cluster-variant of Equation 5 to compare the distance between sibling clusters. Should it be less than a threshold \( d_{sib} \), then a weighted average of the two clusters \( c_1 \) and \( c_2 \) is taken.

- **Concatenation** joins clusters together in the case where a parent node has only one child cluster.

- **Pruning** gives us the option to remove clusters from the tree that have not been recently updated.

The clustering algorithm has been implemented in C++ under ROS.

C. Map Creation

The map creation process is akin to an inverse Occupancy Grid building process, and is outlined in Algorithm 1. It is a fast, online technique. Each time a new cluster tree arrives the existing map is cleared. Each node of a cluster \( c^n \) is a 2D probability distribution \( \mathcal{N}(\hat{c}^n, \sigma^n_{\sigma_x}, \sigma^n_{\sigma_y}) \) describing the likelihood of people traversing the location centered at \( \hat{c}^n \). Clusters that represent popular trajectories will exhibit low variance in the nodes.

We want the people-centric costmap to place low costs on areas of the map commonly traversed, and high costs elsewhere. Essentially, what algorithm 1 implements is half of the occupancy grid mapping process that results in areas in which our observations fall having their costs reduced. We
generate observations by drawing \( n \) samples from each cluster node distribution.

**Algorithm 1 People Map Creation**

```plaintext
for all clusters \( c \) do
    for all nodes in cluster \( n \) do
        Generate \( k \) samples from \( N(\mu_n, \sigma_n) \)
        for all \( s=1 \) to \( k \) do
            \((x_k, y_k) \leftarrow \text{QUANTIZE}(s_x, s_y)\)
            \( \text{Map}[x_k, y_k] \leftarrow \text{Map}[x_k, y_k] + m_{\text{free}} - m_{\text{lo}} \)
        end for
    end for
end for
```

III. EXPERIMENTS, RESULTS AND CONCLUSIONS

Experiments were carried out using a MobileRobots GuiaBot shown in Figure 1. A set of 16 different trajectories were gathered, shown in Figure 3. All trajectories emanate from roughly the same point, an area of 2 metres diameter at the exit to the lift shaft. They are uni-directional, radiating away from this point to 6 different locations on the floor. Although this is a small dataset, our final map in Figure 5 already shows significant areas corresponding to high foot-traffic. Also notable is that the raw dataset comprises 5929 location points, but the final set of 9 clusters shown in figure 4 has a total of 261 points. It is this skeletal representation of the trajectory data that means costmap creation can be done on-the-fly. Figure 6 compares the results of planning paths from the lift exit to 3 different locations on the floor using our people-centric map, and the default navigation stack in ROS that makes use of Occupancy Grids and inflated obstacles, shown in red. There are notable differences in the paths, our costmap has successfully captured the common 'channels' that people walk in on the floor, and this reflected in the plans.

In conclusion, this work shows promising preliminary results in leveraging the powerful skeleton tracking capability of OpenNI and RGBD cameras in order to better navigate human environments.

REFERENCES