Computationally Efficient Path-Following using Adaptive Color Models

Stuart Glaser, Eitan Marder-Eppstein, and William D. Smart
Department of Computer Science and Engineering
Washington University in St. Louis
One Brookings Drive
St. Louis, MO 63130
United States
{sig1,em4,wds}@cse.wustl.edu

Abstract—The ability to follow man-made paths and roads is an important capability for a number of robotic tasks. To operate in outdoor environments designed for humans, autonomous robots must identify footpaths, and drive along them. In this paper, we describe a computationally efficient approach to identifying and following footpaths, using only a single camera. Our technique takes the robot's kinematics into consideration when planning the best trajectory to follow the path that it is on. We show that our approach is highly robust to visual artifacts such as shadows, lighting changes, ground texture changes, and occlusions.

I. INTRODUCTION

The ability to follow man-made paths and roads is an important capability for a number of robotic tasks. Autonomous vehicles must successfully identify the road on which they drive. An outdoor tour-guide robot must identify the footpaths on which it leads its humans. Once the road or footpath is identified, the robot plans and executes a trajectory, attempting to keep on the detected road. Often, these robots have an extensive range of sensors, and make restrictive assumptions about the environment that they will be operating in.

In this paper, we present a robust method for following footpaths using only a single camera, and taking into account the robot’s kinematics. Our system is able to automatically deal with a variety of path materials and textures, changes in lighting conditions, shadows, and occlusions due to pedestrians. We make no a priori assumptions about the path material, or the surrounding environment, relying on an adaptive modeling system to dynamically maintain models of the path in front of the robot. Once areas of path are detected, potential robot trajectories are overlaid on them and scored for suitability. The most suitable trajectory is then executed by the robot.

We are particularly interested in following paths on the campus of Washington University in St. Louis. These paths are typical of a university campus in the United States, and include a variety of construction materials, including concrete, brick, and red clay. The paths are generally bordered with grass, dirt, bushes, or walls. Numerous buildings and trees cast shadows across these paths, considerably altering the perceived color of the path material. The paths also tend to be quite narrow (approximately 1.5m) and winding, often with sharp turns. Additionally, they are often congested by pedestrians. These pedestrians often occlude the path, especially when they stop to stare at the robot. All of these factors combine to make finding the footpath and navigating along it a challenging task. Some representative images of the paths on the Washington University campus are shown in figure 1.

We begin with a review of related work, highlighting the assumptions about the operating environment that are made. We then describe our system in detail, and illustrate how if finds paths and plans robot trajectories. Finally, we give the results of our experiments on the campus of Washington University, discuss the performance of the system, and offer some thoughts about future improvements.

II. RELATED WORK

Much of the work done on path following involves outfitting an automobile to travel along a marked roadway. Several projects use the high frequency of features parallel to the roadway to determine the road shape [1]. Others orient...
by finding road markers [2]. However, footpaths lack the structure of a traditional roadway, therefore, the features that structured road following methods exploit are not present in typical footpaths.

Building on the successes of the methods for following structured roads, researchers began to develop methods for following unstructured roads creating new techniques to tackle the lack of exploitable structure. Such techniques often aimed to model the color and texture characteristics of a road to differentiate it from its surrounding environment [3], [4], [5], [6]. We found color techniques for modeling road appearance useful, but the reliance upon building a precise model of the road structure proved too restrictive for footpaths. Other techniques combine information from multiple sensors, including cameras, laser range finders, and radar devices [7], [8]. Unfortunately, such a variety of sensors is limited for many projects, whether due to cost or on-board computing power. Stanley [7], for example, has five laser sensors, two RADAR sensors, GPS, and uses six on-board computers to perform its computations.

One of the new trends is to combine multiple methods of road finding into a robust framework that outperforms a single method. Though our method only uses information from a single cue, the camera, it can easily be integrated into a method that uses multiple cues [9].

III. ASSUMPTIONS

Before describing our approach, we would like to explicitly set down the assumptions we make about the robot and the environment in which it operates.

1) The robot is on the path. We assume that the robot starts on the path, pointing more-or-less along it. This assumption allows us to assume that the area visible to cameras directly in front of the robot is part of the path. We use this area to initialize our adaptive path modeling system.

2) A large amount of path boundary is visible. We assume that the paths are relatively narrow, and that the camera is positioned in such a way as to see a good deal of path/no-path boundaries. We use this to set an adaptive threshold for the path/no-path classification.

3) The robot moves along circle arcs. We assume that we are using a wheeled robot, and the wheel configuration moves the robot along the arc of a circle, as is the case with differential drive and several other common wheel configurations. We use this to project the path of the robot forward in time, given constant speed and turning rate.

IV. APPROACH

Our method consists of six main parts as shown in figure 2. The frame grabber is responsible for camera control, and captures images for the system. We generate a number of candidate trajectories for the robot, corresponding to circle arcs. These trajectories are mapped in image space, and onto the captured images, by the transformer. These image-space trajectories are evaluated using an adaptive color model and a thresholder. Finally, the trajectory selector picks the best instantaneous trajectory, and passes this on to the drive control which moves the robot. The process then repeats, immediately selecting the next appropriate steering direction. We discuss the details of each of these steps below.

A. Frame Grabber

We use a single color video camera for this work, capturing images with VGA resolution (640 by 480 pixels). The camera is mounted centrally on the robot, pointing slightly downwards so that it can see the path directly in front of the robot. We use Intel’s OpenCV library for our image processing [10].

B. Transformer

We assume that the robot travels along circle arcs, at least over the time taken for a single cycle of our control system. This is a reasonable assumption for robots with differential drive steering systems if we keep the translation and rotation velocities fixed for the duration of a cycle. Each possible trajectory is defined by its curvature \( \kappa \), the inverse of the radius of the circle on which it lies. Points on the trajectory are defined by \((\kappa, s)\), where \( s \) is the distance of the point from the robot along the arc, where \( s = 0 \) is the current location of the robot.

Defining point in terms of \((\kappa, s)\) is convenient because it allows for a straightforward driving model: to drive to a point \((\kappa, s)\), the robot must simply drive along the curve defined by \( \kappa \) for a distance of \( s \). The image-ground transformations, however, are more easily performed in Cartesian space. In order to use the most appropriate coordinate system at each step, we transform between a point \((x, y)\) in Cartesian space and point \((\kappa, s)\) in curvature space:

\[
x = \frac{1}{\kappa} \sin (sk) \\
y = -\frac{1}{\kappa} + \frac{1}{\kappa} \cos (sk)
\]
Fig. 3. An overhead view of the robot in the world, with the field of view of the camera drawn as a trapezoid. Possible driving trajectories are displayed as dotted lines, with the optimal trajectory (the trajectory that remains on the path the longest) highlighted. The robot can check the trajectories where they pass through the field of view to determine how long each stays on the path.

\[
\kappa = -\frac{2y}{x^2 + y^2}, \quad (3)
\]

\[
s = -\frac{x^2 + y^2}{2y} \arcsin \left( -\frac{2xy}{x^2 + y^2} \right). \quad (4)
\]

Figure 3 shows an overhead view of the possible paths the robot can take in the real world. The curves exist in the ground plane, and must be translated to image space. Our cameras are calibrated to the ground plane, and we use a straightforward homography to translate the curves from world space to image space.

C. Color Model

We maintain a color model that scores each image pixel by how likely it is to represent a part of the path. The color model uses a Gaussian mixture model to represent the colors on the path, and updates this model every cycle to accommodate changes in the path color and texture. Our assumption that the robot begins on the path allows us to build the initial model from a pre-specified area of the first image, corresponding to the region of ground (path) directly in front of the robot.

As the robot moves through the world, the mixture model is constantly updated, incorporating information from the same part of the camera image.

The color model maintains a history of path appearances as a mixture of 20 Gaussian, representing the most frequent colors appearing on the path. The history is incrementally updated by combining into the model information gained from a patch of pixels assumed to be on the path, taken from the bottom middle of the camera image. The color model incorporates this new information by using the k-means algorithm to generate a set of two temporary Gaussians from the image patch. These new Gaussians are merged into the history collection, effectively updating the path appearance model similarly to that described by Dahlkamp and colleagues [11].

Once the color model is constructed, it is used to score pixels on the curves projected into the image. The score is computed by taking the minimum Mahalanobis distance, \(d\), between the pixel and all Gaussians in the model. We then assign a score, \(s\), to the pixel according to

\[
s = \exp(-d). \quad (5)
\]

This score represents how well the pixel matches the color of the path, as represented by the current color model. Pixels with higher scores are more likely to represent regions of path in the image.

D. Thresholder

Once the pixels have been scored, we must determine the boundary between the pixels representing regions on and off the path. Determining the boundary is equivalent to thresholding the score, so that all scores above the threshold are most likely on the path. Since the threshold determines whether a pixel corresponds to the path, setting the threshold incorrectly could lead to a poor classification. Furthermore, because the both the color model and the path conditions change regularly, a constant threshold will not suffice.

We chose to dynamically set the threshold based on the scores of the pixels on the trajectories. Because the trajectories begin at the robot, the beginning of each trajectory is certainly on the path. Most trajectories will stray off the sides of the path and remain off the path for a considerable distance. We set the threshold by clustering the scores, leveraging the fact that many pixels on the trajectories are on the path and many are off the path, but few are on the boundary. The scores of the pixels on the path form one distinct cluster, and the scores of the pixels off the path form another. We use the k-means algorithm to cluster the scores into two classes, represented by Gaussians. We estimate both the mean and the variance of these Gaussian clusters.

Figure 4 shows an example of this clustering technique. Scores from the pixels along two different trajectories are plotted on the chart. In both plots, the scores begin relatively high (when the trajectory is on the path), and then drop sharply (when the trajectory leaves the path). The thresholder separates the scores into the two clusters shown on the graph, so that the upper cluster corresponds to pixels on the path and the lower cluster corresponds to points off the path.

E. Trajectory Selector

Once the pixels along each path have been scored and the threshold representing the on-path/off-path boundary has been determined, we select a trajectory to drive along. Since our goal is to remain on the path, the best trajectory is the one that stays on the path for the greatest distance. We step along
Fig. 4. The scores of pixels along two trajectories generated from our data set are plotted. The thresholder performs expectation-maximization on the scores and learns the boundary between the two indicated clusters. The learned boundary clearly separates the scores of pixels on the path from the scores of pixels off the path.

each possible trajectory until it leaves the path, and remains off the path for 30cm. This gives us an on-path distance for each candidate trajectory. We select the trajectory with the greatest on-path distance and send this information to the drive controller.

F. Drive Control

Drive control receives a trajectory to drive along from the curve selector and sets the robot’s heading accordingly. The drive control is also responsible for scaling the robot’s speed based on the curvature of the trajectory it is assigned. For example, as the robot makes a sharp turn, its speed is reduced considerably. This allows for more reliable path-following in the presence of wheel slippage, especially with a skid-steer wheel arrangement.

V. EXPERIMENTAL RESULTS

In this section, we present the results of our experiments with the path-following algorithm described in section IV. The system has been tested extensively on the campus of Washington University and has proven to be extremely robust on all of the public walkways there.

A. Comparison to Human

Developing a good metric for path following performance is difficult, since it is hard to establish what the “right” trajectory is, for any given situation. For this work, we have chosen to compare the trajectories selected by the system to those chosen by humans, looking at the same image sequence.

In order to gather data on human path selections we had five subjects select the best path for each image in a set of 164 images showing paths with varying color and texture. These images were taken from test runs of the system on the Washington University campus. Each human subject was presented with a sequence of images and, for each image, used the mouse to indicate the best driving direction, represented by a trajectory with variable curvature. We compiled the human data by taking the mean and standard deviation of the selected curvatures for each image. The mean represents the average human trajectory choice, and the standard deviation corresponds to the level of ambiguity of the path. A path could be ambiguous, for example, if the path is very wide or if diverging paths exist in the image.

We then ran our algorithm against the same image set and compared its trajectory choice in each image to the choices the humans made. Figure 5 displays the results of a subset of the entire sequence. The points with error bars show the mean and one standard deviation of the curvature of human trajectory selections, and the solid line shows the robot’s choices. The solid line rarely deviates from the human means when the standard deviation is small, and remains nearby when the standard deviation is large. Over the entire image set the robot’s trajectory selection is within 0.5145 standard deviations of the human mean, taking into account the resolution of the robot’s curvatures. Thus, we believe that our approach selects a trajectory comparable to the average human trajectory.

B. Robot Performance

We also tested our algorithm by running it on an iRobot ATRV Junior mobile robot on the Washington University campus. The robot was placed on paths of varying shape and texture including concrete, brick, and red path material. The robot generally ran until the path hit a dead end and rarely left the path.

Figure 6 shows a sample image of the run displaying the trajectory selection process. The algorithm accurately determined where each trajectory left the path and subsequently selected the trajectory that remained on the path the longest.

The algorithm proved robust to both gradual and abrupt changes to the perceived color of the path. Gradual changes,
Fig. 5. A subset of the results comparing human path selection against robot path selection. The points with error bars indicate the mean and standard deviation of human choices for each image in the set, and the line indicates the robot’s selection for the same images. The robot’s selections closely match the humans’ selections, illustrating the power of our method.

such as slight variations in lighting and texture, were handled gracefully by the color model. Abrupt changes, such as shadows, glare, and new path materials, evoked a more complex response. As shown in figure 7(a), when the robot encountered an abrupt change to a previously unobserved color, it was unable to discover a path extending beyond the change. Because it could not find a long path, the robot slowed down resulting in more frequent updates to its color model. Eventually, the model learned the new color, enabling the robot to proceed. As shown in Figure 7(b), the next time the robot encountered a similar color on the path, the model correctly recognized the color, allowing the robot to proceed without reducing speed.

VI. DISCUSSION

Shadows can create a drastic color change on a path surface. There has been a wealth of research on detecting and eliminating shadows from an image [12], [13], [14]. Many solutions to the shadow detection problem could be easily integrated into our method. We anticipated choosing one of these for our implementation, however, in 30 hours of outdoor testing the robot was never driven off the path by a shadow. We attribute this to our adaptive threshold. When the robot finds itself in a position to choose between leaving the path and entering a shadowed portion of the path, the color model gives marginally better scores to the shadowed region. The adaptive threshold is presented with a set of low scores, but it correctly recognizes the distinction between the low shadow scores and the extremely low off-path scores. The robot then turns away from the path edge and into the shadow, and continues along the path. Also, turning into the shadow creates a new history Gaussian in the mixture of Gaussians color model. The new history Gaussian enables the robot to recognize shadows as path regions in the future.

Although we keep a large history for the Gaussian mixture color model we use only two Gaussians to build a color model in each frame. Creating too many Gaussians results in over-fitting the color model, with the “extra” Gaussians modeling colors uncharacteristic of the path. Creating too few Gaussians, on the other hand, results in a poorly modeled path because they cannot handle a path composed of multiple colors. Using two Gaussians in conjunction with a large history has proven to be a good balance.

Our algorithm also focuses on choosing a reasonable trajectory for the robot to drive along and then following it, rather than identifying the precise structure of the path. This stands in contrast to most road following applications which construct a precise model of the road before following it. We found this structure to be unnecessary for footpath following applications. As long as the robot stays on the path, we accomplish our goal. Therefore, a trajectory that matches the model of a path is sufficient for driving.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a novel method for following footpaths using an adaptive color model. The paths that our robot follows are different from roads, necessitating different methods. The paths have no marking, are relatively unstructured, and lack the predictability of roadways. As such, we have had to adapt existing methods and supplement them with new techniques. We have taken methods of following unstructured, unmarked roads, and adapted them to use for path following. We have used a mixture of Gaussians color model to approximate the color distribution of paths. This model is supplemented with an adaptive method for discovering the boundary between pixels on the path and pixels off the path. Finally, we combine the path prediction with a method of selecting the direction by examining only
using edge detection to search for path boundaries. Using multiple methods/cues would ensure that the weaknesses of our method could be compensated for by leaning on different methods.

Our method only determines the steering direction in the immediate future — it does not handle navigation planning. To competently navigate a network of pedestrian paths, an autonomous vehicle must have the ability to remain on the path, to identify and navigate intersections, and to perform higher level planning on the network. Our method handles the problem of remaining on the path for the immediate future, but the other two problems must be handled by different modules. Once methods for handling intersections and for higher level planning are developed, they can be integrated with our method to create a robotic system capable of autonomously navigating a network of footpaths.

**ACKNOWLEDGMENT**

Special thanks to Joe Izraelevitz for his contributions to this work.

**REFERENCES**


