IP05: Tracking Multiple RC Cars

Mark Woodward, Stanford University

Abstract

In this paper I present a method for tracking a single primary remote control car and locating other cars nearby using a single pan-tilt-zoom camera. The method uses color filtering to track the primary car, and optical flow and particle filters to locate nearby cars.

Introduction

This paper presents a method for tracking a single primary remote control car on a racetrack, and for locating nearby cars. The method presented here utilizes a single pan tilt zoom camera.

Previous systems for tracking objects in a laboratory environment fall short of what is presented here. They either, use a single stationary camera and thus cannot cover as large of an area with the same accuracy, or they utilize multiple cameras and require more extensive setup.

Tracking of the primary car was done by applying a color filter to the image, finding the point at the center of the matching pixels, rotating the point into the original extrinsic frame, and finally intersecting the point with the ground plane.

In a parallel thread, the car is kept in view by commanding the camera to look at the center of the filtered color.

Location of nearby cars was done with optical flow to find regions with high magnitudes of change, and particle filters to track these regions. Particles were weighted based on their distance from flow vectors and their distance from other filters.

Estimates of locations are produced by averaging the x and y locations for particles of a given filter, and then rotating and projecting those estimates onto the ground plane.

Approach

The problem of tracking robots in the lab environment has been studied in great detail by the robotics community. My work was carried out because none of the commercial systems available fit my needs. Specifically, I needed a system that was easy to set up, and that could track an RC Car as it roamed over a 40' x 20' area, with 10' high ceilings, to an accuracy of 2".

Systems that use a single stationary camera were ruled out because of the limitations of the height of the ceiling. If placed directly overhead, the coverage area for a standard webcam was only a 5' x 5' patch. If the single camera was placed to one side, at a sufficient angle to cover the entire area, the perspective effects reduced the accuracy, preventing the desired 2" accuracy.

Using a fish eye lens produces a combination of these bad effects. Coverage area is still not what I need, and distortion reduces the accuracy.

Other systems use multiple cameras to cover the entire workspace with high accuracy, see Mezzanine (1) developed at USC. These systems do not fit my application because the multiple cameras are not easy to set up. My specific application, namely tracking remote control cars during a race, requires quick setup and take down, since preparation time before races is limited.

A system using a single pan tilt zoom camera meets all the above requirements. It can be quickly setup at a racetrack, using a tripod, and can accurately track a car as it roams over a 40' x 20' area.

The approach I developed solves two main problems. The first is tracking a primary car on the racetrack. The second is locating other remote control cars in the vicinity of the primary car.

Tracking Primary Car

The primary car is tracked by applying a color filter to the video received from the pan-tilt-zoom camera, locating the center of the matching pixels, rotating the resulting vector to the frame in which the extrinsics were originally computed, intersecting the vector with the ground plane to find the world coordinates, and finally, in a separate thread, commanding the camera so that the primary car remains in the center of the view.

A color filter is used because it provides a simple, accurate technique for locating an object when the color of the object is known a priori, and when that color is not found in large quantities in the viewing environment. If in practice, other cars are painted with the same color as the primary car, markers could be used. But this would result in greater processing time.

The filter simple thresholds the blue/red ratio, the blue/green ratio, and the blue intensity for a given pixel. If the rgb values for a pixel are greater than the three thresholds, then the pixel is a match.

A sum of the x coordinates for all matching pixels is kept, along with a sum for the y coordinates. At the end of the single pass, the location of the car is,

x = sum_x_coords/num_matching_pixels
y = sum_y_coords/num_matching_pixels

Figure 1 shows this filter applied to a given image. The white pixels are the pixels that satisfy the filter. The red crosshairs are drawn at the computed location of the car.

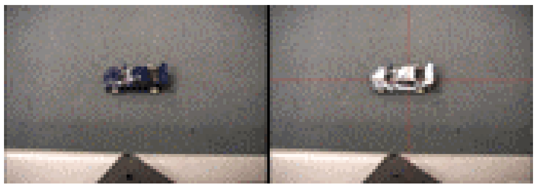
Figure 1. Before and after color finding

The location of the car defines a vector in the current camera orientation. To find the location of the car in world coordinates, we need to intersect this vector with the ground plane of the racetrack. But, the extrinsic for the ground plane were likely computed in another orientation of the camera. So, we have to rotate the vector into the original frame

P_car_in_orig_frame =
R_y(pan)*R_x(tilt)*P_car_in_current_frame

The vector for the point is then rotated into the world coordinates.

P_car_in_world_frame =
R_cam_to_world_frame*P_car_in_orig_frame

Where R_cam_to_world_frame is found from the original extrinsic camera calibration.

The intersection of the car vector with the ground plane is found by solving the following for t, x, and y

P_cam_in_world_frame +
t*P_car_in_world_frame = [x, y, 0]^T

This gives the location of the car in world coordinates. Without further action, the car would quickly move out of the field of view of the camera, and be lost.

To keep the car in view, my approach uses a separate thread that constantly monitors the P_car_in_current_frame vector, which is an output of the previous thread. Given the angle of view of the camera, the thread computes the pan_error and tilt_error. These are the angles required to bring P_car_in_current_frame in line with the vector defined by the principle point. Finally, the thread commands the pan-tilt-zoom camera by these angles. This loop is executed as fast as possible, which in the tested camera case is about 4Hz.

Figure 2 shows two consecutive frames, as the camera centers itself on a car that has stopped.

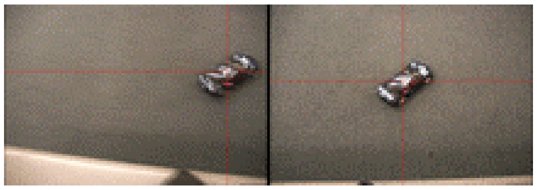
Figure 2. Pan to center car

Given a known color, the system can accurately track a car over the desired area and to the desired accuracy.

Locating Nearby Cars

The second problem addressed is the location of remote control cars that are nearby, in this case so that the client application can control the primary car to avoid them.

The solution presented here is to use optical flow to find features that have moved a lot in consecutive images, and maintain a particle filter per car, whose particles are weighted by their proximity to optical flow vectors and by their distance from other filters.

The optical flow field is computed using the Shi and Tomasi algorithm to find features to track, and then the Pyramidal Lucas Kanade Optical Flow algorithm to compute optical flow vectors for those features.

Optical flow was used because of its ability to capture magnitude, and because of its ability to quickly pull out the objects of motion in the image. As seen in figure 3, once the proper threshold was applied, all flow vectors refer to points on the car.

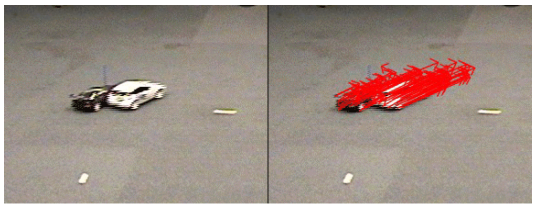
Figure 3. Before and after optical flow

An alternative to optical flow might be frame differencing. But frame differencing does not as easily avail itself to magnitude estimates, which when thresholded significantly reduce the number of features. After the frame differencing, the application would be left with deciding which pixels are relevant.

The optical flow vectors are then used in the weighting step of the particle filters.

The technique presented here uses at least one particle filter per vehicle in the race. Since the number of vehicles in a race is always known before the race begins, this can be hard coded.

At any given time, multiple filters may latch on to a single car, but as long as all cars in the given view, which approximates the area near the car, contain at least one filter then all obstacles to avoid have been detected.

In the update weights step, for each particle, in each particle filter, the nearest optical flow vector is found. The particle is weighted higher the closer the optical flow vector is.

Also, the particle is weighted higher the further it is from the center of all previous filters that have been processed. This creates the repulsive force, which results in covering all cars in the view.

After all particles in a given filter have been weighted, all flow vectors used by any particle in that filter are flagged, so that they cannot be used by future filters in the iteration. This further servers to spread out the particle filters.

During the prediction step of each filter, particles jump to a random location with a probability of .05. This allows the filters to quickly acquire cars that come into the scene.

The right image of Figure 4 shows two filters locked on to two different vehicles. The particles that have jumped to random locations can also be seen sprinkled throughout the image.

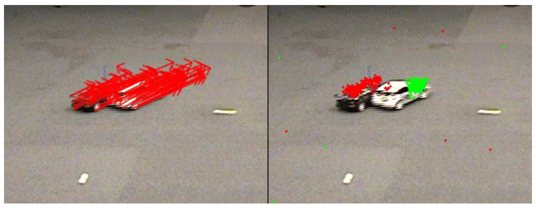
Figure 4. Before and after particle filters

When a client application requires the position of each car in the image, the particle filters need to be collapsed to specific estimates. This is done using an average. The x and y components of each particle contribute to the x and y sum for that filter respectively. Then the sums are divided by the number of particles.

Figure 5 shows the collapsing of two filters using this technique. The green circles in the right image are centered on the vehicle locations.

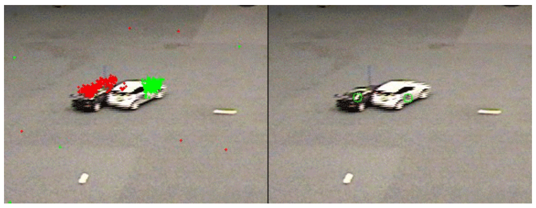
Figure 5. Before and after collapse.

Averaging was chosen because of its accuracy/speed trade off. Only a single pass was required for each filter.

A faster method is to randomly pick one of the particles. While on average, this would give accurate readings, occasionally it would randomly pick particles that had jumped to random locations, producing large jumps in the location of the vehicle.

A more accurate estimate would have been the location of the median particle. But, this would have required the sorting of all particles in the x and y directions, which is much more processing intensive than the average technique used here.

The averaging of each filter results in car locations given in the current camera frame coordinates. The collapse routine then projects the computed locations onto the ground plane in the same manner as was done for the location of the primary car, taking into account the current pose of the camera.

These steps result in world coordinate estimates for all cars near the primary car.

Incidentally, the optical flow generated by the primary car will also attract a particle filter and thus give a location estimate. The client application can easily throw these estimates out by removing all estimates that are within a radius that corresponds to the radius of the primary car.

Results

The camera used to test the technique is the Sony EVI-D30 (figure 6). It is capable of 12 times optical zoom, 200 degrees pan at a rate of 80 degrees per second, 50 degrees tilt at a rate of 50 degrees per second, and is controllable via a serial port connection.

This camera was chosen because of its high pan and tilt speed, and because the command protocol is published, allowing me to write my own drivers.

Figure 6. The Sony EVI-D30 PTZ Camera

The RC car used for testing is the XRAY M18 (figure 7). It was chosen because of its small size and because it has an active racing community in the San Francisco bay area.

Figure 7. The XRAY M18 RC Car

Here I present several image sequences that illustrate the success of the proposed technique.

Tracking of primary car

The next set of images show the camera actively tracking the car. The track lags behind the center of the car due to time delays in commanding the camera. The first frame is the bottom left.

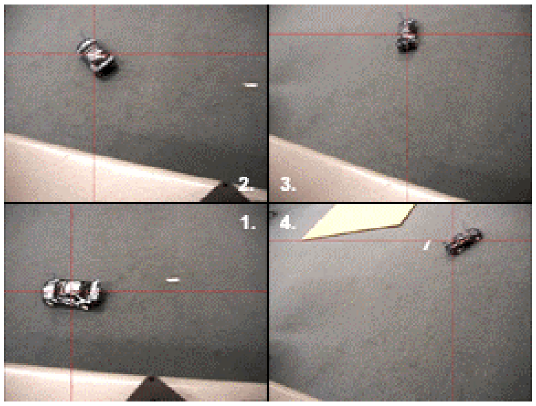

Figure 8. Tracking the primary car. Bottom left, clockwise. 1 second per image.

Location of nearby cars

Figure 9 shows a series of consecutive optical flow images. It's clear how nicely the flow vectors stick to the vehicles. The displayed vectors are the result of the thresholding, which is hard coded.

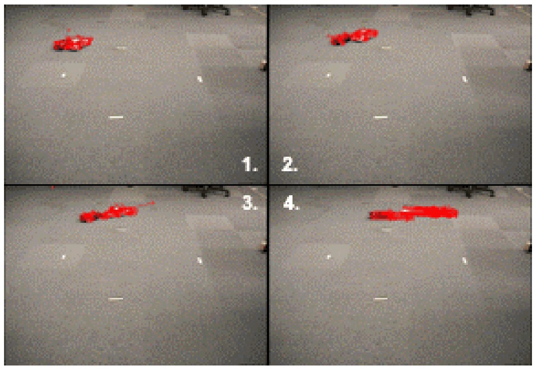

Figure 9. Optical flow for two cars. Left to right, top to bottom. Every 10th image is displayed.

Figure 10 shows the corresponding particle filters for the same images.

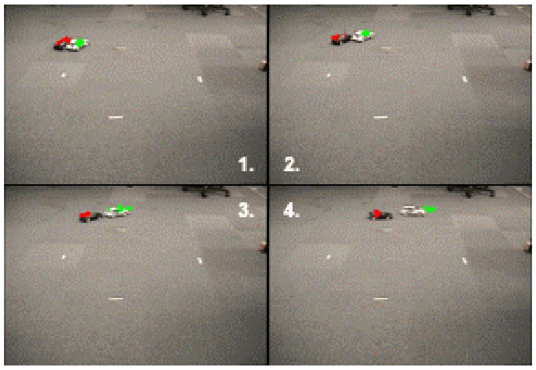

Figure 10. Particle filters for two cars. Left to right, top to bottom. Every 10th image is displayed.

Figure 11 shows the first two images collapsed down to estimates. Estimates are shown as green circles.

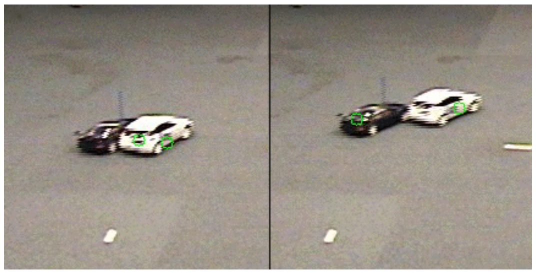
Figure 11. Position estimates for two frames 10 frames apart from each other.

Figure 12 shows a series of images in which, originally two particle filters are locked on to a single car, but as a second car comes into view, one filter quickly transitions to it.

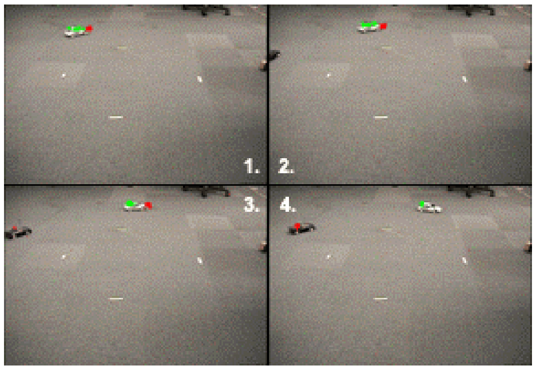

Figure 12. Acquisition of new car in view.

Summary

This paper presented a technique for accurately tracking a primary remote control car operating on a 40' x 20' track using a single pan-tilt-zoom camera, and providing estimates for nearby rc cars.

Color filtering was used to track the primary car, and the resulting vector was rotated and projected onto the ground plane. Finally the camera was controlled to keep the primary car in the center of view.

Optical flow and particle filters were used to track nearby cars. Particles were weighted based on their distance from flow vectors and their distance from other filters. Filters were collapsed to estimates using the mean of the particles. Finally, image estimates were rotated and then projected onto the ground plane to give world estimates for all nearby cars.

Results show that the primary car can be accurately tracked using the techniques presented. Also, it was shown that all cars in the field of view are quickly and accurately tracked using the techniques presented.

An interesting alternative to pursue would be to place markers all over the track and then use these markers to estimate the pose of the camera for each frame received. This would eliminate the small inaccuracies experienced during the panning and tilting of the camera. The downsides to this technique would be more image processing and added setup time to place the markers around the track.

References

1) Howard, A., "Mezzanine User Manual", USC Robotics Laboratory
2) Kato, H., Billinghurst, M. (1999) Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). October, San Francisco, USA