Joe Rollo
Advisor: Prof. Reid Simmons
Undergraduate Senior Research Thesis

Tracking for a Roboceptionist

Abstract

Currently, the roboceptionist's ability to decide whether or not to greet a nearby person is inadequate. Often the roboceptionist attempts to greet people who are not interested in talking back, but merely passing by. The roboceptionist may also attempt to greet people who are standing outside the nearby classroom—these people are not interested in responding either. While the roboceptionist is attempting to greet people who will not respond, other people who do want to talk to the roboceptionist may be waiting for a greeting. The problem is that the roboceptionist is spending too much attention on people who are not interested in interacting with the roboceptionist.

Before the roboceptionist can reliably greet people, however, the roboceptionist needs to have a reliable way of tracking people. This project refines a velocity-based motion model for particle-filter based person tracking using a laser range finder.

I. Introduction

Robust person tracking is an important part of human-robot interaction. In order for a robot to make predictions about human behavior, that robot must have a reliable set of data from which to make predictions. One recurring problem in person-tracking is the problem of "target discontinuity": the tracker may lose track of a person due to an occlusion, or even a brief failure in data association. If the person reappears, the tracker assigns a new ID to that person, and disregards that person's previous tracking information. We can improve the tracker's motion model so that when a temporary loss

of data occurs, the tracker uses its current estimate of the person's position and velocity to predict the subsequent location of the person.

When people walk normally, they do not always walk in exactly the same direction as they were walking before. Some trackers model this trait by using Brownian motion [1, 8]. According to the Brownian motion model, a person is equally likely to move in any direction at any moment. In reality, however, the chance that a person suddenly moves backward is much less than the chance that a person continues to move forward. So Brownian motion is a crude model for the motion of a walking person. We can improve our motion model by biasing Brownian motion in the direction that a person is moving. Now our motion model captures both the person's tendency to deviate from their forward direction and the fact that people generally favor the directions that are closer to moving forward.

Before we can use a person's velocity to bias our motion model, the tracker must be able to track a person's velocity. Unfortunately, laser range scans at the thigh level can result in noisy velocity data. We explore this problem by trying various filtering methods to track person velocities.

We will explain all of the steps in this approach: segmentation of laser data into people blobs, assigning filters to people blobs, tracking position with a particle filter, tracking velocities, and incorporating a velocity bias into a Brownian motion model. We will also verify the improved tracking reliability with results based on laser range scans of people walking near the roboceptionist.

II. Related Work

Many contributions have been made to the field of people tracking using laser range finders. One approach to tracking is to use a Kalman filter [5]; this approach assumes that the state being estimated changes according to a Gaussian distribution. A non-Gaussian approach is to use a particle filter [3]. This approach represents the state of a moving target with a set of random samples. This approach works best when the samples according to a motion model that accurately predicts how the people being tracked walk through the environment.

Some people particle filter-based tracking methods use a Brownian motion model [1, 8] in order to avoid tracking person velocities. The imprecise nature of this model becomes apparent whenever observations of a person are not available. During an occlusion, or a failure to identify a laser segment as a person, Brownian motion model will cause the particle filter's particles to disperse randomly. Even if the person reappears within the large region of dispersed particles, it is often the case that there are not enough particles near the person to justify the filter's support for that person. So the tracker assigns a new particle filter to the person, and that filter lasts until the next loss of observation. The Brownian motion model inhibits our ability to produce robust tracking data.

Others have extended the particle filter approach using sample-based joint probabilistic data association [6, 7]. While this approach offers a more sophisticated method of associating features in laser data to objects being tracked, the motion model is restricted to a Gaussian distribution over changes in walking direction and speed.

Another technique involves learning common destinations in the environment, and then incorporating these destinations into a goal-based motion model [2]. This approach relies on a body of training data in order to work. Not only must the data be large enough to cover all the paths that people might follow; this data must also be free of occlusions if the trainer uses a Brownian motion model. Furthermore, a trainer that uses a Brownian motion model would need a perfectly reliable algorithm for associating laser segments with people; otherwise, brief failures in data association would cause the trainer to have tracking discontinuities. The plan-based tracker would need an excessively large amount of laser data in order to avoid such errors. In this case, the cost of the Brownian motion model is in the required amount of training data.

III. Tracking Using Particle Filters

A. Using a Laser Range Finder to Observe People.

Before we can track a person, we need to know what a person looks like. Our tracker is based on a SICK laser range finder, so our robot's perception of the environment is a series of 2D laser scans. So a tracker update begins with a time-stamped laser scan. The tracker's first step is to create a list of observed person locations based on the current laser scan. This algorithm was already provided when we began this project [8].

The laser range finder has a 180 degree range, and a resolution of 1 laser beam per degree. This sensor gives the tracker an array of distances and positions, sorted by the angle at which each laser beam is fired. We partition the set of laser beams into

segments by imposing a threshold for how close two consecutive laser beam positions must be in order to belong to the same segment. We define the center of a laser segment as the centroid of the positions given by the laser beams in that segment. We define the width of a segment as the position difference vector (w_x, w_y) between the first and last laser beam in the segment, where the "first" and "last" refer to indices in the array of laser beam data. This gives us a set of laser segments, each with a center and a width.

For each laser segment, we impose a set of fixed thresholds on the magnitude of the segment's width to determine whether a laser segment refers to a person. We also have fixed thresholds for identifying legs, since it is possible to perceive a pair of legs as two separate segments. If a segment does not conform to either of these thresholds, then we label that segment as a wall, which we ignore.

For leg-matching, we use a nearest-neighbor algorithm, and we have thresholds on distance between leg centers to determine which legs are close enough to be considered a pair. Whenever we match a pair of legs, we merge the leg segments and redefine the new pair's center and width. Since an unmatched leg is not enough information to track a person reliably, the tracker ignores unmatched legs.

Now we have a segmentation of our laser data into "people blobs", where each blob is an observed position of a person. These blobs define what a person looks like to our tracker.

B. Assigning Filters to Observed People

This step in our tracker assigns each person blob a filter for tracking. Initially, we have no filters, so we simply create a new filter for each person blob. This process gets

more complicated, however, when we must match already existing filters to new observations. Our person tracker solves this problem using a nearest-neighbor approach. As with the method from part (A), this algorithm was already provided when we began this project [8].

First, we calculate the centroid of each particle filter (if any exist at the current iteration of the tracker); here, we define the centroid as the mean of the filter's sample positions. Then, for each person blob, we compute the distance from that blob's center to the centroid of each filter, and take the minimum of these distances. If the minimum distance is within a constant predefined threshold, then we assign the closest filter to the person blob. Otherwise, none of the filters is close enough to the person blob for us to justify an assignment of an existing filter to that person blob. Note that if the filter that we just assigned turns out to be closer to another person blob, then we will reassign that filter to the closer person blob.

This process may leave some person blobs unassigned to a filter. In this case, we instantiate a new filter and assign that filter to the person blob. Also, some filters may not be assigned to a person blob, due to an occlusion, a blob segmentation error, or a person leaving the environment. In case the person still exists undetected in the environment, we allow a filter to remain in an "unassigned" state for a predefined constant amount of time before we remove the filter. This removal timeout resets when we assign a person blob to a previously unassigned filter.

In some cases, a filter that was previously assigned to a person blob may become too far away from the person blob for reassignment, due to inability to track the person blob. When this occurs, the person blob is assigned a new filter, and old filter becomes

unassigned—destined for premature removal. As a result, the person blob changes filter ID's and the tracker discards all of the old filter's data. This is the *tracking discontinuity* problem, which a robust tracker should avoid as much as possible. In part (IV), we base our tests for tracker robustness on the occurrences of these discontinuities.

Now that each person blob has an assigned filter, we can use these filters to track the person blobs.

C. Particle Filters

Our tracker uses particle filters [3] in order to assign each observed person an ID that persists through consecutive laser scans. This approach uses random samples and sample weighting to estimate the location of a person. We use a Gaussian distribution with the person's location as the mean to generate the filter's first set of particles. Once we have a set of random particles for our filter, we update the particle filter in two steps: move the particles according to a motion model, and resample according to particle importance weights.

First, we move the particles according to some motion model. The goal of this motion model is to predict where the next location of the person will be as accurately as possible. Some person-trackers use a Brownian motion model [1, 8] to move the samples. A goal-based motion model [2] is also possible. In part (E), we will introduce a motion model that uses both current velocity and Brownian motion to move the particle filter's samples.

We then resample our particles based on "importance". Here, importance is defined by proximity to the center of the person blob that the particle filter is tracking.

The following is a formal definition of proximity-based importance weight. Let (x_i, y_i) be the location of the particle filter's *i*th sample; let (x_c, y_c) be the location of the person blob; let w_x and w_y be the width of the person blob in the x and y direction respectively; define Gaussian_probability(x, Φ, σ) as the Gaussian probability distribution function with mean Φ and variance σ. Then we compute the ith sample's importance weight P with

$$\text{gauss\_x} = \text{Gaussian\_probability}(x_i, x_c, \sqrt{\frac{w_x}{2}}) \quad \text{(C1)}$$

$$\text{gauss\_y} = \text{Gaussian\_probability}(y_i, y_c, \sqrt{\frac{w_y}{2}}) \quad \text{(C2)}$$

$$P_i = \text{gauss\_x} * \text{gauss\_y} \quad \text{(C3)}$$

After calculating an importance weight for each sample, we then use the importance weights to create a new sample. We do this by creating a random sample selector with probability distribution function *p* over our old sample set (of size *n*) such that

$$p(\text{select the } i\text{th sample}) = \frac{P_i}{\sum_{j=1}^{n} P_j} \quad \text{(C4)}$$

We then use this random sample generator repeatedly until we have a set of *n* random samples. Since the random sample generator favors samples with larger weights, our new sample set will have samples with larger importance. Since importance is based on proximity to the person blob, our new sample set shifts closer to the person blob as a result of resampling.

By moving the samples according to a motion model and resampling based on importance weights, we can track the location of a person blob as the person moves through the roboceptionist's field of view.

D. Tracking Velocity

At this point, we have everything we need in order to implement a tracker that uses a Brownian motion based particle filter. However, a tracker that incorporates velocity into the motion model also needs a method for estimating a person's velocity. Simply taking the difference between current and previous blob centroids is inadequate, because this leads to imprecise velocities—from now on, we will refer to this notion of velocity as "raw velocity". The noisiness of raw velocity comes from leg movements that distort person blobs. Since our laser range finder is aimed horizontally at the height of most people's legs, human gaits have a confounding effect on the velocities of person blob centroids. Nevertheless, we can remove some of the noise by smoothing the raw velocity values with a filter (see Figures 3-1 and 3-2). This brings us to the question of which filtering algorithm is appropriate for smoothing velocity.

Although a simple average-based filter may not be optimal, this method works well-enough to mention as a possibility. Two particular kinds of average-based filter are the weighted average, and the moving average.

Weighted Average: let Z_{n+1} be the current raw velocity, V_n be the previous weighted average velocity (or the first raw velocity, Z_1, for n=0), and α be a predefined constant in the range [0,1]. Then we compute the current weighted average velocity V_{n+1} with

$$V_{n+1} = \alpha Z_{n+1} + (1-\alpha)V_n \quad \text{(D1)}$$

This involves finding a "good" value for α, which is possible via trial and error. A lower value of α puts more emphasis on the previous weighted average; this "dampens" the velocity values.

Moving Average: let k be the number of consecutive raw velocity values to average together, where $k \neq 0$. We compute moving average velocity V_{n+1} with

$$V_{n+1} = \frac{1}{k}\sum_{i=0}^{k-1} Z_{n+1-i} \quad \text{(D2)}$$

This gives us the sum of the current raw velocity (where $i = 0$) and the k-1 most recent previous raw velocities. A higher value of k puts less weight on the current raw velocity, thus "dampening" the velocity values. We can find a reasonable value for k using trial and error.

Another approach to velocity filtering would be to use a discrete Kalman filter [4]. This filter is a feedback control system that updates both the state $\hat{x}$ and the error covariance P of a system. There are two steps in a Kalman filter update: a time update, and a measurement update.

The time update step predicts what the state will be in the next time step. We update our predicted (i.e., a priori) state and error covariance with

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \quad \text{(D3)}$$

$$P_k^- = AP_{k-1}A^T + Q \quad \text{(D4)}$$

where $\hat{x}_k^-$ is the predicted state, and P_k^- is the predicted error covariance. In (D4), Q is the process noise covariance, which we define as a constant. Here, $\hat{x}_{k-1}$ is our previous state value, which contains the previous filtered velocity V_n. In the context of the velocity filter, $u_{k-1} = 0$ because we have no control input for a person blob's velocities.

Given a predicted state and error covariance, we use an actual measurement to correct our prediction in the measurement update step. We correct our prediction with

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad \text{(D5)}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad \text{(D6)}$$

$$P_k = (I - K_k H)P_k^- \quad \text{(D7)}$$

In (D5), R is the measurement noise covariance, which we define as a constant. The key feature of this step is the computation of the Kalman gain matrix K_k. The equation (D5) computes K_k so that the velocity estimate error covariance (i.e., a posteriori error covariance) P_k is minimized. We then plug the Kalman gain K_k, the measurement z_k (which is the same as our raw current velocity Z_{n+1}), and our predicted values $\hat{x}_k^-$ and P_k^- into equations (D6) and (D7). This gives us a new state estimate $\hat{x}_k$ and a new error covariance measurement P_k. Here, $\hat{x}_k$ contains our current filtered velocity V_{n+1}. So the measurement update gives us our filtered velocity value, as well as the input values for the next Kalman filter update.

In part IV, we will measure the performance of the biased Brownian motion person tracker using each of the velocity filters mentioned above. Figures 3-1 and 3-2 demonstrate the noise removal of the weighted average velocity filter. Figures 3-3 and 3-4 compare three velocity filters mentioned above. Note that the velocity vectors are

filtered component-wise in terms of x-component and y-component, and then converted into direction and speed for the graphs in figures 3-1 through 3-4.

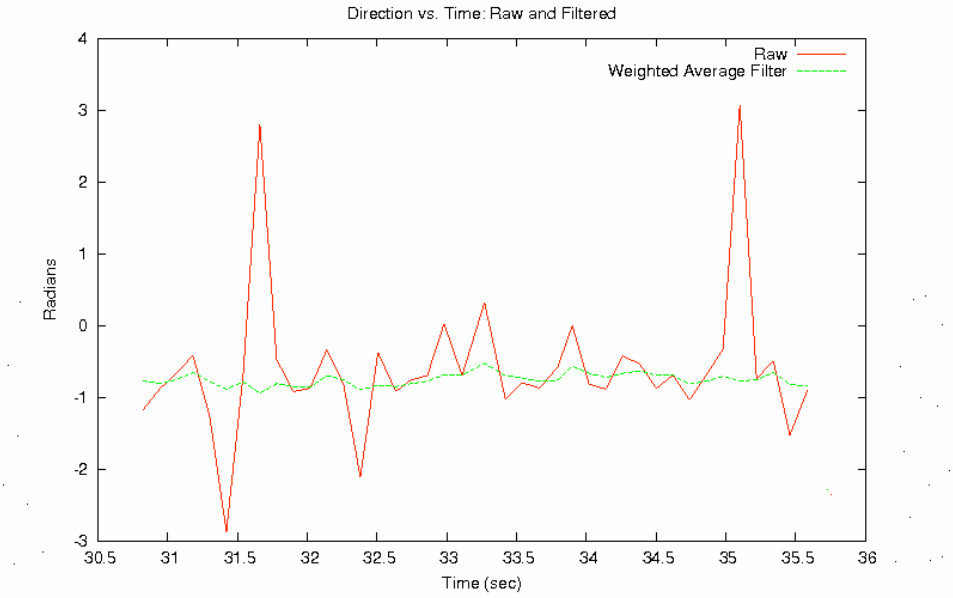

Figure 3-1: A weighted average filter removes noise from a person blob's estimated velocities over time. This graph shows the direction component of the "raw" and filtered velocities.

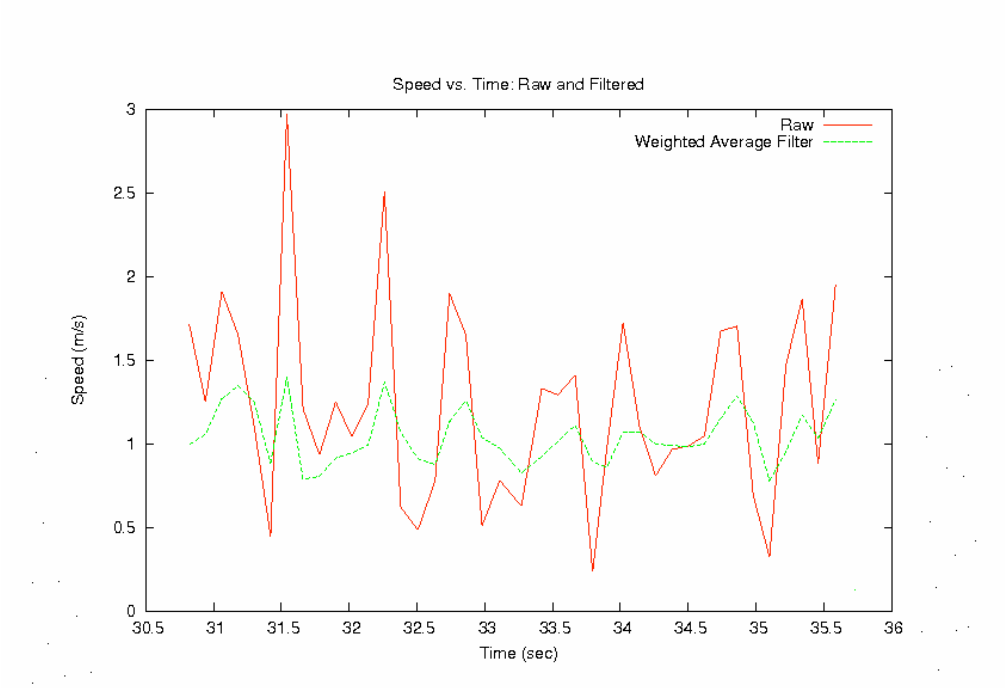

Figure 3-2: A weighted average filter removes noise from a person blob's estimated velocities over time. This graph shows the speed component of the "raw" and filtered velocities.

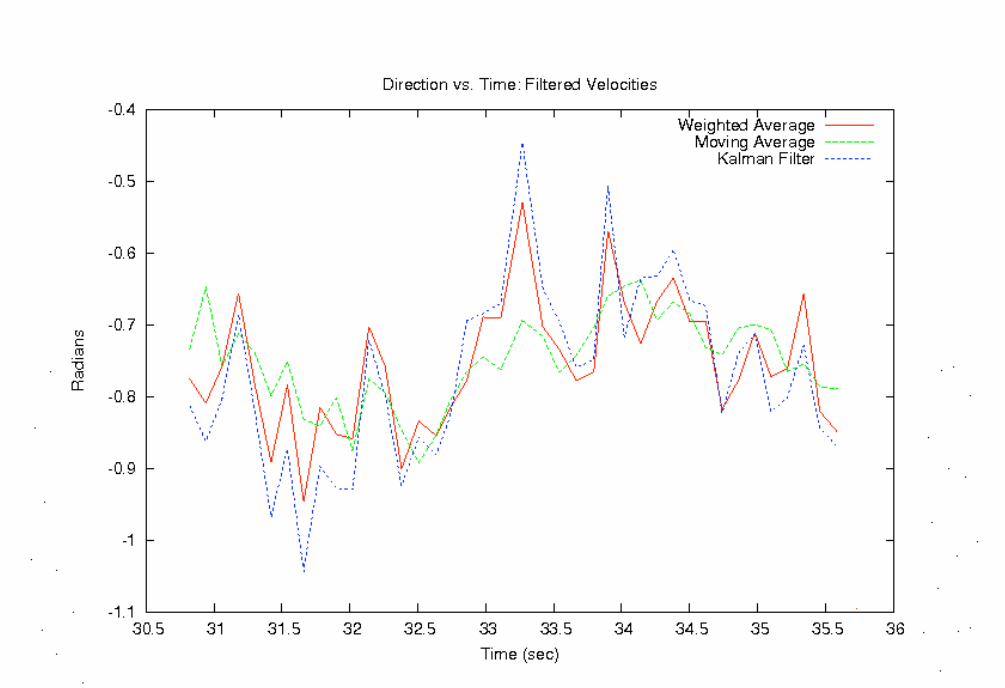

Figure 3-3: Comparison of three different velocity filters. This graph shows the direction component of the filtered velocities.

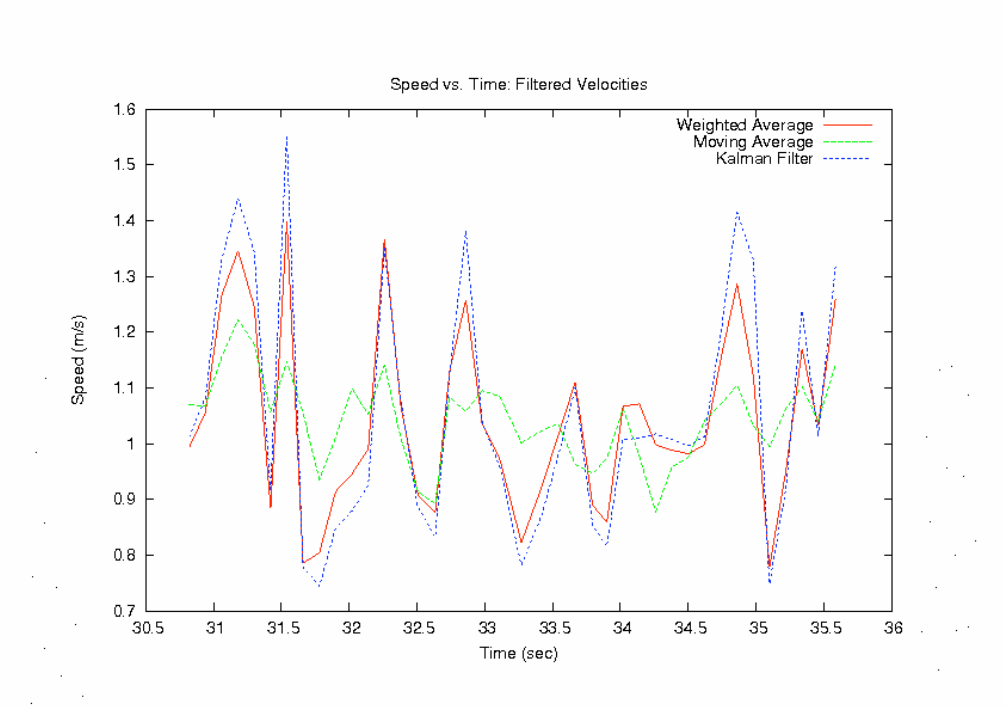

Figure 3-4: Comparison of three different velocity filters. This graph shows the speed component of the filtered velocities.

E. A Biased Brownian Motion Model

With a velocity filter in place, we can incorporate person velocity into the Brownian motion model. This results in a motion model that still assumes random movement, but biases the movement with the person's average velocity.

Brownian motion models use a two-dimensional Gaussian distribution with each particle's position P as the mean and a fixed constant σ for the variance. So a person tracker that uses a Brownian motion model would use the following formula to update the position P of each sample:

$$P_{new} = \text{Gaussian}(P, \sigma) \qquad \text{(E1)}$$

In order to bias this motion model, let Δt be the time elapsed since the previous tracker iteration; let V_n be the estimate of the person's current velocity. Now we can replace (E1) with

$$P_{new} = \text{Gaussian}(P + V_n \Delta t, \sigma) \qquad \text{(E2)}$$

Note that by translating the mean of the normal distribution in the direction of the person's estimated velocity, we are incorporating both the person's direction and speed into the motion model.

Intuitively, the model that uses (E1) expects the person to wander randomly at the particle filter's current position, while the model based on (E2) expects the person to wander randomly from the next point on the person's path as projected by the person's current velocity. The biased Brownian motion model (E2) makes a stronger assumption than (E1) about where a person is likely to move, but still allows the person to deviate from the expected trajectory.

IV. Results

We constructed three tracking tests for our person tracker. All of these tests use real laser data taken from Tank the roboceptionist's SICK laser range finder. We stored the laser data into log files so that we could replay the exact same laser data for each variant of our person tracker. Each test compared a Brownian motion person tracker to three biased Brownian motion person trackers which vary by velocity filter; the velocity filters we used were a weighted average, a moving average, and a discrete Kalman filter.

For the weighted average filter, $\alpha = 0.25$; for the moving average filter, $k = 11$; for the discrete Kalman filter, $Q = 1$ and $R = 500$.

We hand-picked all of these velocity filter parameters using a simple trial and error technique. First, we define upper and lower bounds in filtering, where an upper bound refers to a filter that filters velocity too heavily for good tracking, and a lower bound refers to a filter that puts too much "trust" in the raw velocities. For the average based techniques, it was easy to distinguish upper and lower bounds by inspection. An excessively "heavy" filter would cause the particle filter to "fall behind" a person blob as a result of being too slow to respond to changes in speed and direction. With a lower bound, on the other hand, filtered velocities would have excessively large magnitudes, causing the particle filter to "fly off" the person blob. With these distinctions in mind, we managed to close the bounds enough so that all values within a certain range produced roughly equivalent results. For the weighted average filter, the range for α is roughly [0.1, 0.4], where 0.4 is the "lower bound" as defined above. For the moving average, the range for k is roughly [6, 16], where 6 is the "lower bound". Unfortunately, the distinctions between "lower bound" and "upper bound" were less clear when tuning the Kalman filter parameters; we had difficulty noticing any difference in performance for R in the range [300, 700], so our choice for Q and R may be somewhat crude.

A. Counting Discontinuities

The first test measures the average number of "tracking discontinuities" (as defined in part III-B) per person tracked.

This test uses a continuous 214-second interval of logged laser data. I hand-counted 21 moving objects (which we will assume to be distinct people) as I inspected this data segment. Throughout this interval, people walk through the roboceptionist's field of view. We chose this particular data segment because there are few occlusions, and all of the occlusions are brief—so we can ignore the effect of occlusions in this experiment. Also, the people in this data segment walk in relatively straight paths; this gives us a simple test for tracking people who walk in straight paths.

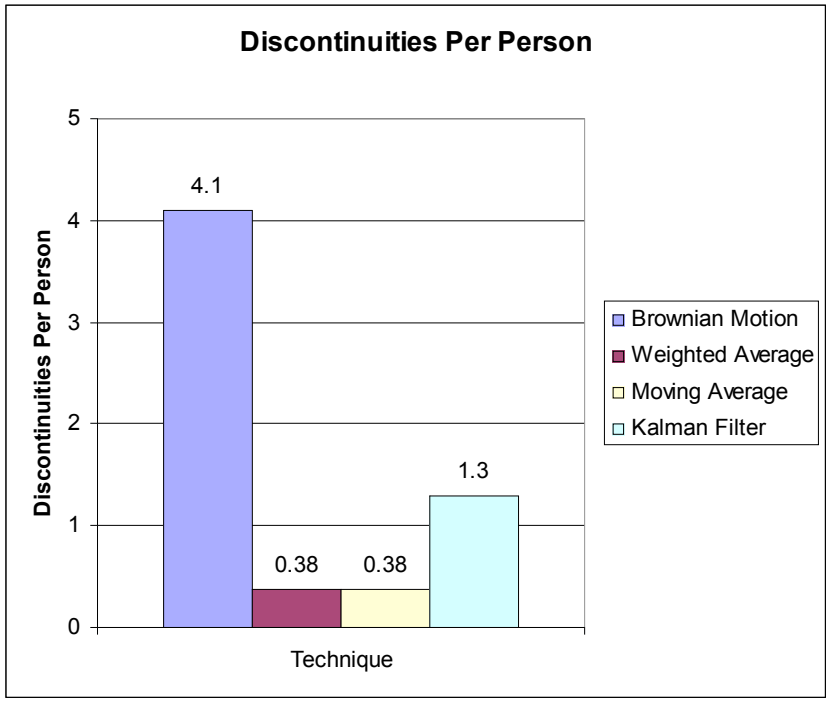

Figure 4-1: The average number of discontinuities per person, based on 21 people.

Dividing the number of discontinuities for each technique by the number of people tracked (21) gives us Figure 4-1. Our Brownian motion person tracker has an average of over 4 mistakes per person—an alarming error rate for a person tracker that is supposed to be reliable. The suboptimal performance of the Kalman filter is due to a

crude choice of parameters Q and R (hand-picked for our convenience)—nevertheless, using a Kalman filter on velocities in this way was still better than using a Brownian motion model. The biased Brownian motion person trackers that use an averaging technique have less than one mistake per person; while this is an improvement, these results suggest that over a third of all people may be tracked incorrectly.

B. Measuring the Success of a Person Tracker

Although the results from part (A) show improvement in tracking with biased Brownian motion, they fail to capture the full extent of our method's success. It is unfair to define success completely in terms of the number of tracking discontinuities, because these errors may not be evenly distributed over the duration of a person's traversal through the robot's field of view. Consider the first one-second time interval after a new person blob appears. This appearance is likely to occur near an edge of the robot's field of view, where laser data is slightly less reliable and velocity estimates noisier. A velocity filter initially magnifies this noise, since velocity estimates do not instantaneously converge to a smooth value. Therefore, we can expect our biased Brownian motion filters to make a significant number of mistakes during the first second of tracking. Once a velocity filter becomes "stable", however, we expect biased Brownian motion to be more robust than Brownian motion.

To accommodate this velocity filter "set-up time", we can relax our definition of tracking success so that if the tracker stops making errors after a certain initial period, then the tracker is "successful". In the context of the roboceptionist, we can afford to

wait one second after a person appears before the tracker stops making initial tracking errors.

Our second test measures the success of our person tracker with our new, more relaxed definition of success in mind. This test uses the same laser data that our test from part (A) used. Instead of counting the number of discontinuities, we count the number of "successfully tracked" people. Here a person is "successfully tracked" if the tracker makes no mistakes after 20 tracker update iterations (roughly one second) from the initial appearance of that person in the roboceptionist's field of view.

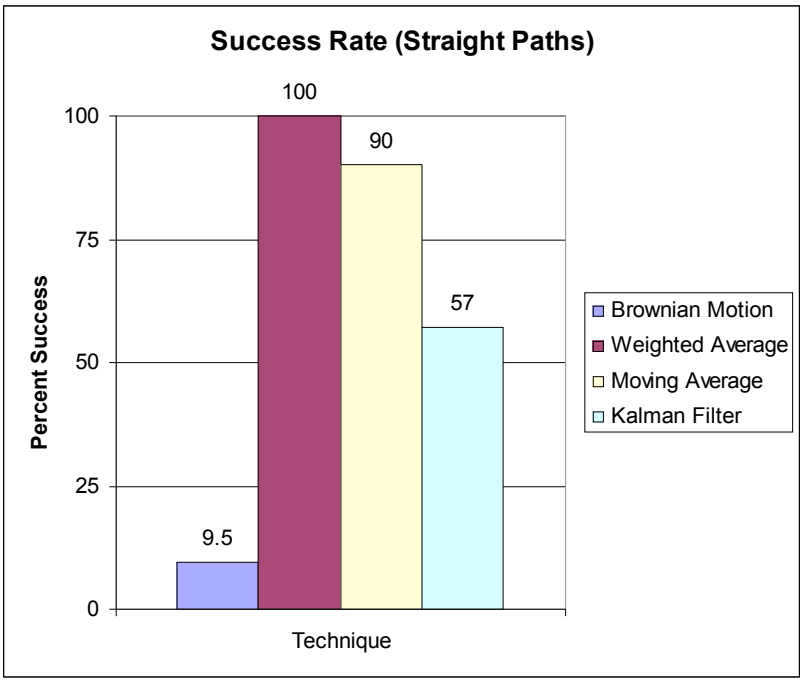

Figure 4-2: Tracking success rates, based on a relaxed definition of success. "Straight Paths" refers to the 21 people who traversed the roboceptionist's field of view during the test

Figure 4-2 shows the results of this test. As illustrated in Figure 4-3, the Brownian motion person tracker still performed poorly due to tracking discontinuities throughout a person's path. The weighted average and moving average person trackers had high success rates—in these cases, most (or all) of the errors occurred within the first second of tracking a person. The Kalman filter still shows mediocre results due to a crude choice of parameters Q and R.

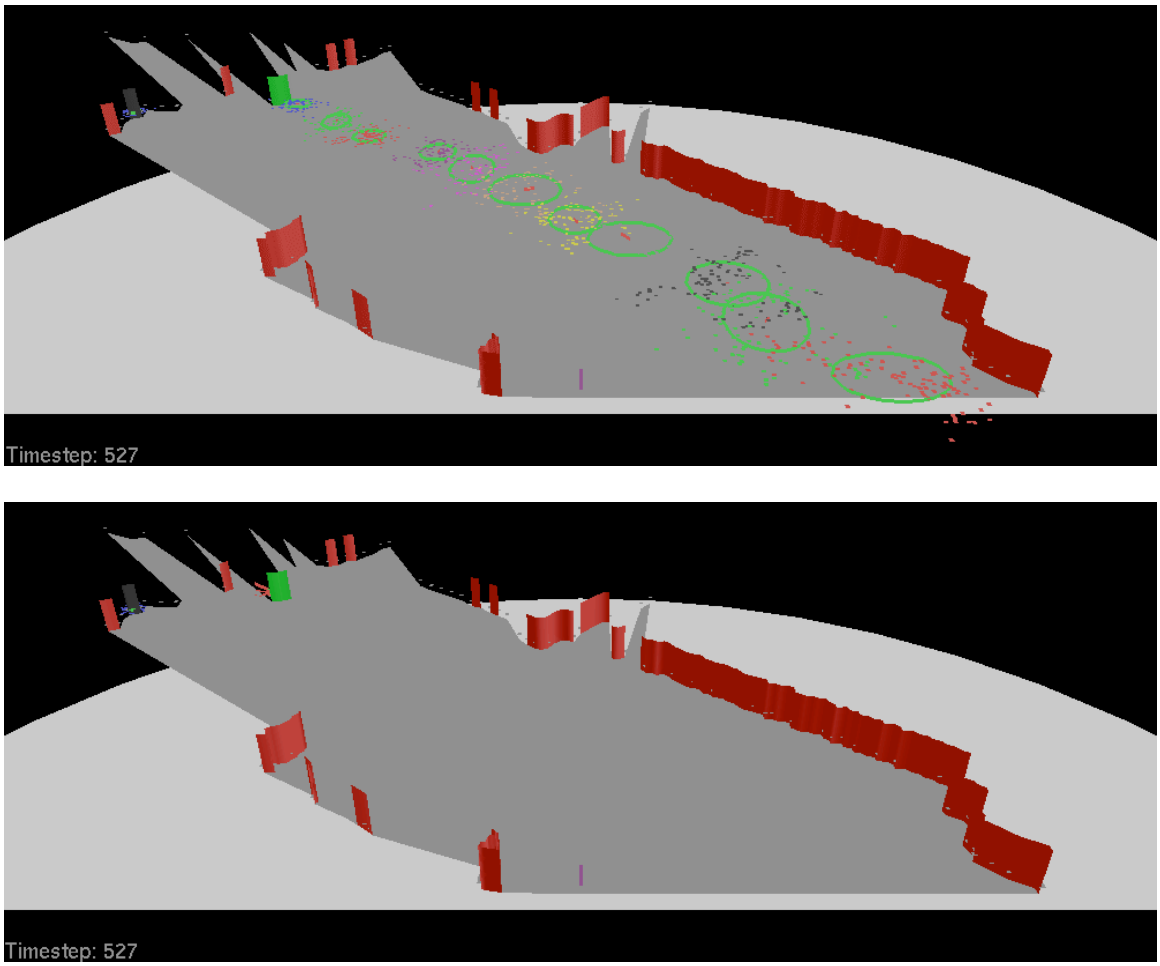

Figure 4-3: Screenshots of the "Straight Paths" test in progress. Top: using a Brownian motion model. Bottom: using a biased Brownian motion model with a weighted average velocity filter. The circles with dots are particle filters. The ribbons are laser segments (green ribbon means person blob, red ribbon means wall).

C Tracking With Occlusions

The results from part (B) measure performance for occlusion-free paths. In the real-world, however, occlusions can have a major impact on person tracker performance. So we devised an experiment in order to measure how robust a person tracker is to occlusions.

This test uses a continuous 48-second interval of logged laser data. Throughout this interval, a stationary object in front of the roboceptionist's sensor casts a shadow of occlusion down the center of the roboceptionist's field of view. I hand-counted 19 moving objects (which we will assume to be distinct people) passing through this shadow of occlusion as I inspected this data segment. So this test gives our person tracker nineteen chances to track an occluded person successfully.

For this test, a person is considered "successfully tracked" if that person keeps the same particle filter before and after walking through the stationary shadow of occlusion.

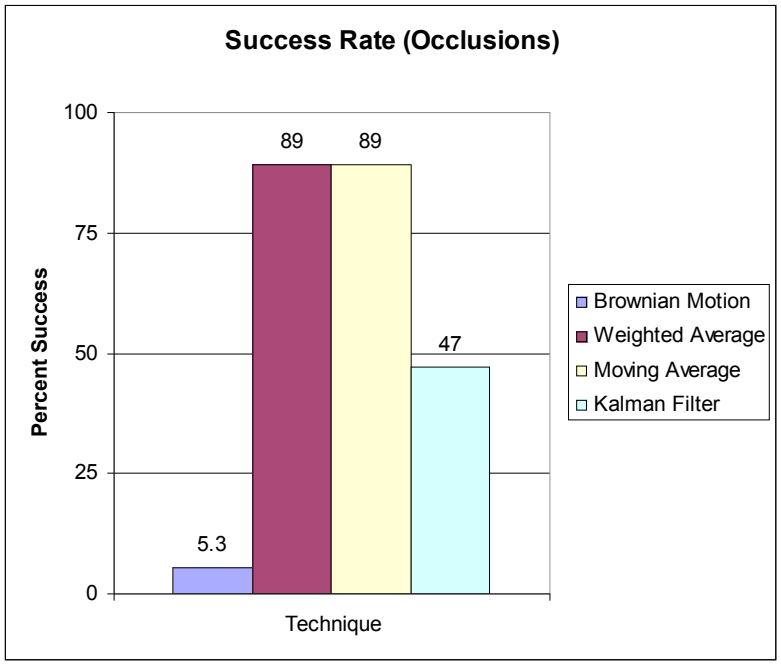

Figure 1-4: The rate at which a person is successfully tracked despite walking through a stationary occlusion. This is based on 19 people walking through the same shadow of occlusion.

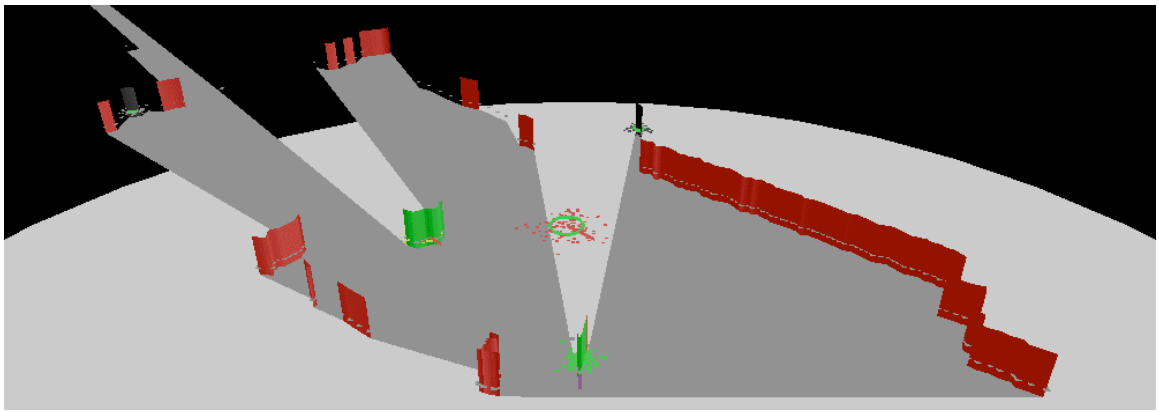

Figure 2-5: Occlusion test screenshot. This shows a stationary target occluding a person, and another person approaching from the left. The ribbons are laser segments (green ribbon means person blob, red ribbon means wall). The dots in the shadow of occlusion are the samples of the filter that is temporarily "unassigned" during the occlusion.

Figure 4-4 shows the occlusion test's results, and figure 4-5 shows a screenshot of the test in progress. Although the weighted average and moving average person trackers were not perfect, they performed far better than the Brownian motion person tracker.

Even the person tracker that used a Kalman filter for velocity values, despite crudely chosen parameters Q and R, performed better than the person tracker that used a Brownian motion model.

V. Conclusion

In this paper, we presented a method for incorporating both velocity and Brownian motion into a person tracker's motion model. We showed that by filtering velocity values and biasing the Brownian motion in the direction of current velocity, we can improve the reliability of our person tracker. Our results demonstrated improved robustness in person tracking, even in the midst of occlusions.

As a bonus, our method tracks velocity as well as position. A robot that uses our person tracker would be able to consider a person's position and velocity when deciding how to interact with that person. A roboceptionist, for example, would be able to ignore a person who is nearby, but walking quickly past the roboceptionist.

Our person tracker is far from perfect, however. Simply ignoring the first one-second time interval is a significant cost for person trackers that need to respond as quickly as possible to a newly-detected person. It would be better solve the "set-up time" problem, or at least reduce the set-up time need for reliable velocity tracking. Also, it might be possible to improve tracking velocities with a Kalman filter by using a system identification technique [4]. Another area in need of improvement is our blob-segmentation algorithm, which relies too heavily on fixed thresholds; this method has edge-cases in which people are temporarily mistaken for walls. As far as the question of which velocity filter to use, there are many other possible filters beyond the three that we

chose to test. We mentioned one way to bias a particle filter's motion model, but there could be other more advanced methods of bias: consider biasing both the mean and the variance of the Gaussian distribution at each step. These are but a few of the imperfections in our person tracker.

One major flaw in our method is our assumption that the velocity of a person is equivalent to the velocity of a person blob's centroid. This assumption does not hold true because we are tracking pairs of legs, not torsos. An understanding of how leg movement corresponds to torso movement would greatly improve our estimations of person velocities. This would improve the robustness of our person tracker.

Acknowledgements

Thanks to Rachel Gockley and Frank Broz for starter code, in particular the person blob segmentation and the assignment of filters to person blobs. Thanks to Marek Michalowski for Kalman filter starter code.

References

[1] M. Montemerlo, S. Thrun and W. Whittaker, "Conditional Particle Filters for Simultaneous Mobile Robot Localization and People-Tracking," in *IEEE International Conference on Robotics and Automation (ICRA),* 2002. [Online]. Available: http://ieeexplore.ieee.org/iel5/7916/21826/01013439.pdf?arnumber=1013439

[2] A. Bruce and G. Gordon, "Better Motion Prediction for People-tracking," in *IEEE International Conference on Robotics & Automation (ICRA),* 2004. [Online]. Available: http://citeseer.ist.psu.edu/668426.html

[3] M. S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," in *IEEE Transactions on Signal Processing*, Vol. 50, No. 2, Feb. 2002. [Online]. Available: ftp://ftp.irisa.fr/local/aspi/legland/ref/arulampalam02a.pdf

[4] G. Welch and G. Bishop, "An Introduction to the Kalman Filter." [Online]. Available: http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html

[5] A. Fod, A. Howard and M. J. Matarić, "A Laser-Based People Tracker," in *IEEE International Conference on Robotics and Automation (IRCA),* 2002. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1013691

[6] D. Schulz, W. Bugard, D. Fox and A. B. Cremers, "Tracking Multiple Moving Targets with a Mobile Robot using Particle Filters and Statistical Data Association," in *IEEE International Conference on Robotics and Automation (ICRA),* 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=932850

[7] E. A. Topp and H. I. Christensen, "Tracking for Following and Passing Persons" in *IEEE Intelligent Robots and Systems (IROS),* 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1544961

[8] R. Gockley, J. Forlizzi and R. Simmons, "Natural Person-Following Behavior for Social Robots," in *IEEE International Conference on Human-Robot Interaction (HRI*), 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1228720