Person Following On STAIR: Fall Report for CS221 & CS229

Carl Case
cbcase@stanford.edu

Abstract

In this report I present the results of my work this fall, with the tremendous help and support of Olga Russakovsky, on the STAIR Person Following project. There are, roughly speaking, three components of this project: the vision system; the navigation system; and the recovery system. I describe first the use of the Phasespace Motion Capture System to accomplish the vision task and a simple model of the cameras that allows for quick recalibration. Second, I present the bulk of the work from this quarter: updating the navigation system on STAIR to both integrate with Phasespace and support the all-new ROS navigation stack, and the difficulties concomitant with this task. Finally, I describe early work on the recovery system and present a simple framework in which to apply learning to improve recovery performance.

1 Introduction

While much research has focused on unaided robot navigation in both indoor and outdoor environments, from the robot user's point-of-view, this sophistication may be too much. In particular, specifying a particular goal may be more than the user wants or needs to do. In this situation, it is desirable that the robot be able to autonomously follow the user around, whether there is a specific destination or not.

In person following, as in many other robotic tasks, the two primary concerns are safety and robust performance. The former is of greatest importance: there must be no failure mode in which the following robot puts any person (the one being followed, or otherwise), objects in the environment, or itself in harm's way. With that condition met, we expect the robot to perform as robustly as possible in a dynamic, changing environment. The person should be able to walk at a range of both distances and speeds. Navigable obstacles should pose no problem. And crucially, the robot must recover from any reasonable temporary occlusion or disappearance of the target person.

The goal of this project is to enable the STAIR robot to execute person following pursuant to these specifications. Additionally, we make the following assumptions. First, the person may be slightly "modified" – in particular, we will place a small, distinctive LED on her person. Second, for reasons of safety, we assume a reasonable maximum speed. And third, we assume that the robot will operate in an indoor environment for which it has access to a static (though possibly imperfect) map – generated either from CAD models or mapping techniques.

I decompose the person following application as follows. First, there is a vision system that is responsible for observing and tracking the target person so that the robot may know her location relative to its. Second is the navigation system which converts the output of the vision system to a sequence of velocity commands for the mobile base, enabling following behavior. Third is the

recovery system, which takes over from the vision system when the person goes out of view, providing the navigation system with the information it needs to execute a search to recover following behavior. Note, of course, the imperfect modularity of this decomposition, as there are feedback loops between each of the three systems.

The rest of this report proceeds as follows. First, I will provide a brief description of the vision system as implemented with the Phasespace Motion Capture System. I expect that this system is very near to its final form, and as it is mostly prior work, the overview will be brief. Second, I will describe with some detail the navigation system as it is currently implemented in the ROS framework. Finally, I will present the preliminary status at the moment of the recovery system and plans to improve upon it once (nearly) all bugs are out of navigation.

2 Vision with Phasespace

The hardware for our vision system is two Phasespace Motion Capture cameras mounted on the left and right sides of STAIR. Attached to the person is a small, red LED which pulses in a high-frequency pattern uniquely identifiable by the Phasespace cameras. The two cameras are chained together and communicate with the Phasespace base computer over ethernet. At present, STAIR needs to have a long ethernet cable trail it so the base computer can remain immobile. In the future, the computer will be onboard STAIR, allowing for full autonomy.

Given the accuracy of the Phasespace cameras (the documentation claims sub-centimeter accuracy, and my experience is that claim is very near the truth), the goal is to extract an (r, θ) pair that provides ground truth about the person's location relative to STAIR's.

Phasespace provides software for doing all triangulation calculations automatically to provide distance measures directly in meters. In extensive testing, however, I found that their software is ill-suited for the restrictive setup of two closely-spaced camaras; if the LED went more than two meters beyond the cameras, no distance measures were returned.

As a result, the first part of the project was to implement my own triangulation system layered on top of the Phasespace API which provides access to its raw data. Each camera provides two data points from each of its two linear detectors in the range $[-1, 1]$. The mapping from these 4-tuples to (r, θ) is not immediately obvious, so the first implementation of the triangulation system was based on locally-weighted linear regression, where each $x^{(i)}$ is a 4-tuple of the four raw data values from Phasespace and the target variable is the (r, θ) ordered pair.

Soon after, however, after futzing with the regression for a while and studying the way the camera's values worked, I discovered a simple reverse-engineering of the setup that allowed an analytic solution for both r and θ as a function of any three of the four detector values (recall each camera has two). For reference, if $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})$, then it turns out that:

$$x_1^{(i)} + x_3^{(i)} = C - (x_2^{(i)} + x_4^{(i)})$$
$$x_1^{(i)} + x_4^{(i)} = D - (x_2^{(i)} + x_3^{(i)})$$

where C and D are constants that can be directly taken from training data (in each case, the average distance between the particular pair-wise sum). Let $\alpha^{(i)} = x_1^{(i)} + x_3^{(i)}$ and $\beta^{(i)} = x_1^{(i)} + x_4^{(i)}$. Then:

$$r = \tan(a\alpha + b)$$
$$\theta = c\beta + d$$

Where the parameters a, b, c, d must be learned from data. This is, of course, a straightforward linear regression problem, so I implemented a system that requires the user only to hold one LED at a specified sequence of locations – (x, y) coordinates relative to the cameras, as that is easier to measure than (r, θ) – and outputs the four parameters for directed implementation of the triangulation system. To be overly explicit, solving for a and b requires solving the following optimization:

$$\arg\min_{a,b} \sum_i \left(\alpha^{(i)} - \frac{\tan^{-1}(r) - b}{a} \right)^2$$

and of course the corresponding equation for c and d. In practice, these values can be directly computed using the normal equations.

The software implementation of the vision system pipeline is entirely in ROS, which will merit further discussion below.

3 Navigation in ROS

3.1 ROS

All of the software on STAIR runs on top of ROS, the Robot Operating System[3]. I have little to add here, but it is worth emphasizing the computation model of ROS. Specifically, any application is run as a network (directed graph) of individual nodes which can both subscribe to and publish data messages (the subscriptions / publications forming edges in the graph). As a result, the model enforces modularity between different subsystems, so much of the implementation is a direct mapping of the separate systems I describe here onto individual nodes.

3.2 Person Following and Navigation

There are, I think, a handful of different ways to pose the "following problem," but I have so far settled on the most straightforward. At any point in time, we know the location of the target person relative to STAIR's location. Thus the goal of navigation will be to execute velocity commands that moves STAIR toward the person as quickly and directly as possible while remaining in safe operation. The constraint on safe operation, in general, is obstacle avoidance that takes into consideration the limited dynamics of the robot.

The algorithmic approach we have taken to executing velocity commands is known as the "dynamic window approach" and is described in [1]. This selection was made both because it meets all the constraints mentioned above and because much of the implementation already exists in the core ROS navigation stack. In many ways, this code base ended up both a blessing and a curse as described below, though the balance most definitely falls on the positive side of the scale.

There is no reason, of course, to provide a full descricption of DWA, but I will provide a brief overview to render clear the most relevant details for this project. DWA can be best understood as a search algorithm *in velocity space*. In particular, it first constructs a feasible and "safe" search space parametrized by (v, ω), the linear and angular velocities respectively, and then optimizes an objective function within that search space to make the "best" possible progress toward the goal. The search space is limited by considering only:

1. Circular trajectories – this is the mathematical result of choosing to parametrize velocites as a constant pair (v, ω)

2. Admissible velocities – a velocity is admissible if after choosing this velocity, the robot can stop clear of the nearest obstacle on the resultant path

3. Velocities in the dynamic window – a velocity is in the dynamic window only if it is immediately accessible to the robot given current velocities and torque restrictions

With this set of restrictions and a discretization of (v, ω), the algorithm has a finite space from which to select its velocity commands. This is done by forward simulation: simulate each choice from the search space and score the results by a combination of: proximity to goal; distance from obstacles; and speed (the relative weighting is controlled by user-set parameters). See [1] for a complete description of how to select the admissible velocities, etc.

In some sense, that is all there is to following: specify the visible location of the person as the goal and let DWA handle the rest. Of course, the devil is in the details. In particular, implicit in the description of DWA is the concept of some map to answer questions about proximity to obstacles. There is no reason, in principle, that you could not use a static global map – load it up from an image file and just use that. In practice, however, this depends on two factors on which we cannot depend: excellent robot localization and a near-static environment with respect to the map. If localization is off or there are obstacles not present on the map, then the forward simulation does little to ensure a safe trajectory.

For this reason, the implementation of DWA maintains a local window / map that is always centered around the robot with no reference to global coordinates. This window is six meters a side, discretized into sub-sqaures 25cm a side. Each cell can be in one of three states: occupied, free, or unknown (which are in practice treated as occupied).

The states of the cells depend on the SickLMS laser scanner attached to STAIR. It repeatedly makes 180 degree scans, returning the distance measured at each degree. For all hits less than MAX_RANGE (a user-specified constant), the angle and distance allows you to immediately mark the resultant cell as occupied. Note that the converse operation, clearing cells based on MAX_RANGE hits, is rather more complex, as it involves ray-tracing from STAIR along the line defined by that hit's angle and clearing all cells en route.

Armed with this local map, the implementation of DWA is mostly straightforward, as all forward simulations can be run in the local window without reference to a global map. And if the goal is anywhere outside of the local map, it can be clipped onto the edge and moved as the map moves with the robot.

4 Global Mapping and Recovery

4.1 The Global Map

As simple as the local map makes the following problem when the person is in view, it is insufficient for executing a global search to recover when the person leaves the robot's view. Any search that aims to do something more sophisticated than "go to the last visible goal point and spin in place" must have a concept of where the person might have gone *from there* – which is a question about the global space.

Our approach, then, to global mapping is as follows. As long as the person is in view, the goal is to localize STAIR as best as possible – which is an entirely passive process. If x_t is the state (pose) at time t, z_t is the laser scan measurement at time t, and u_t is the odometric control between $t-1$ and t, then the localization problem is to maintain a distribution $P(x_t|x_{0:t-1}, z_{0:t-1}, u_{0:t})$ where the

subscript ranges are over all values in that range. This can, of course, be formalized as an HMM in the standard way and be reduced with Bayes rule to the state transition probabilities $P(x_t|x_{t-1}, u_t)$ and the measurement probabilities $P(z_t|x_t)$. The former are derived from odometry readings from STAIR's base, while the latter come from the SickLMS laser scanner along with a static map of the environment.

As long as the person is in view, this system plays an entirely passive role, maintaining a representation of the probability distribution over STAIR's pose. In particular, the distribution is maintained with a particle filter and the Augmented Monte Carlo Localization algorithm (amcl). I will elide the details here – for reference, see [4].

4.2 Representing the Person's Location

It should be said, of course, that we are only interested in STAIR's global location on the map to the extent that it allows us to determine where the followed person has gone when she leaves view. The most direct representation is as a single (x, y) point on the map. In particular, given a particle particle cloud representation of the robot's location, we can calculate the mean of that cloud and adjust by (r, θ) as provided by the vision system. As soon as the person leaves view, we can assume that the person must not be too far from the last seen point and navigate directly there.

As a naive strategy, this is not half bad (and the basic approach of the current implementation). It is reasonable to expect that the person will not have gone that far, and a simple navigate and spin strategy can relocate the person in many cases. The difficulty, however, is representational. In particular, it is unclear how to model the likely continued movement of the person if she is represented by a single point. Or more problematically, at a T-junction in a hallway (for example), how to model the possibility that the person will travel in either direction (other than just checking both).

For these reasons, which are basically reasons of robustness, the chosen strategy is to represent the person's location by a probability distribution – just as STAIR's is represented. Specifically, we would like a joint distribution over the pose of both the robot and the person to be followed. There is a good treatment of this problem in Mike Montemerlo's paper [2], in which he presents the Conditional Particle Filter. To represent the joint distribution, the state x_t now consists of both the robot's pose and the person's. Since, however, the person's observed pose is dependent on our belief about the robot's, this probability factors to:

$$P(x_t|z_t, u_t) = P(r_t|z_t, u_t)P(y_t|r_t, z_t, u_t)$$

where r_t and y_t are the robot's and person's poses respectively. The important question, then, is how to represent the conditional distribution $P(y_t|r_t, z_t, u_t)$. This depends on our sensor model. If the *person* sensor is noisy, then that distribution is a proper distribution that must have full representational power; in [2] this is accomplished by associating an entire particle set to represent the person's pose for *each* of the particles in the robot's distribution.

If, however, the person sensor is perfectly noise-free, then that distribution is entirely characterized by the prior $P(r_t|z_t, u_t)$. Merely propagating forward the robot's distribution by the camera measurement of (r, θ) will give you an accurate distribution over y_t.

It is this approach I have taken so far in the early implementation until further testing is possible.

4.3 Recovery Actions

The final piece of the puzzle is how to search for a lost person given the representation we have. Presently, the distribution over the person's pose is collapsed to its mean and STAIR simply navigates to that point and begins spinning – a strategy exactly equivalent to the naive strategy described in the previous section. Future work, then, (beyond making the nav stack less flaky) is to improve upon this strategy.

The next step, I believe, is to allow the particles in the person's distribution to move autonomously as soon as the person leaves view. A first attempt would look something like this. Each particle takes a Brownian random walk from its starting position. Any particle that remains in the visible field of STAIR for a period of time (as estimated from its own pose estimate) is removed from the distribution – as are any that enter a static obstacle. At corners and other junctions with only one possibility, this should have the same effect as "pushing" the distribution around the corner – the only possibility if the person has left view. More interesting are junctions with multiple possible paths. The proposed approach will, I hope, have the effect of generating a multi-modal distribution over those choices. A k-means clustering of the resultant distribution, for example, should generate a set of proposed locations the person might have gone. A set of geometric heuristics can then be applied to those proposed points to create a search ordering. For example, if one is in a room (bounded) and the other is in a hallway (unbounded), we may want to check the latter first, as the person is liable to be lost further only in the hallway.

It is worth noting that these geometric heuristics are one piece of the larger system most susceptible to learning techniques. In particular, the ranking of locations according to "time urgency" can be posed as a classification problem. With a training set that consists of hand-labels for a set of points on the map, just extract features from a local window around those points to train a classifier. This is, of course, future work.

5 Results and Discussion

5.1 Navigation and STAIR

As a preface to any discussion, I feel obliged to explain the exact nature of the project during this quarter in particular. In particular, I acknowledge that much of this report is speculative and preliminary, with less in the way of concrete results (other than a chunk of code) than I would like.

Early in the quarter, after we finished the vision system milestone report, I elected to update the navigation stack on STAIR to an all-new code base that represented a fairly dramatic shift from the original system, with new message types, different nodes, and a reorganized API. Going forward, it seemed worthwhile to move over to this new nav stack, as the original was not entirely dependable and there seemed little reason to develop over a system that was already obsolete. In practice, this turned into much more of a fiasco than I expected – and I discovered how wanting I am in experience debugging complex hardware-dependent systems.

Thus the bulk of the work this quarter does not lend itself to a final report. The odometry and laser systems have been updated under the hood to work with the new system (and remove a few latent bugs). The segway no longer non-deterministically fails to close its port. There is a bundle of new configuration files that describe how STAIR interacts with the new nav stack. And so on.

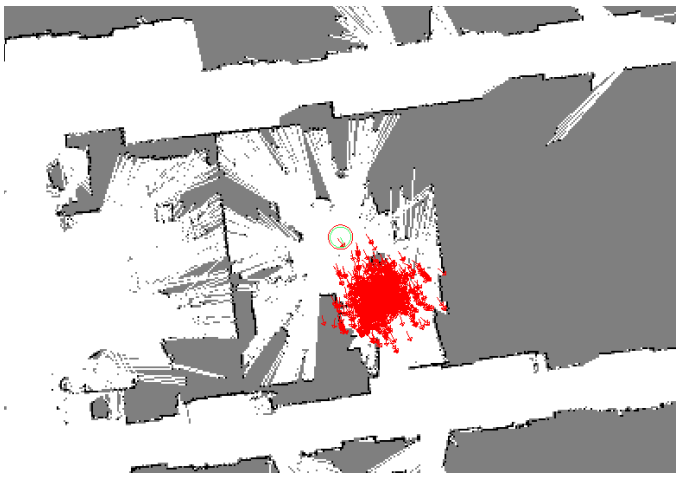
Embarrassingly, there are still a few bugs in the navigation system (it really likes to hug walls as it navigates). The result is that many of the novel, interesting pieces of the actual person following

application are still in their natal stages.

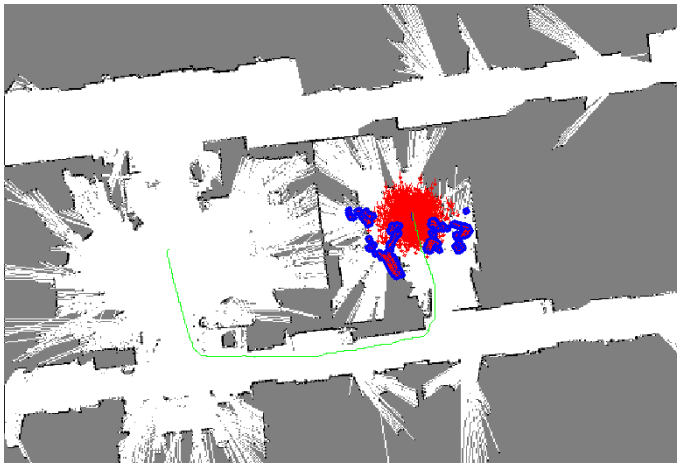
5.2 Results, Briefly

The results and accuracy of the camera system was presented most recently in the October Report for the project, so there is little reason to take space reproducing them in detail here. For reference, in the range of 1-5.75 meters, the Phasespace system is accurate to within 1.5cm. It can still procude results, those less reliable, up to 6.5m. The horizontal field of view is 28 degrees at 1.5m, expanding to 33 degrees past 3m.

Disappointingly, the navigation system is *still* not reliable enough to test out any person search execution without endangering STAIR and the walls of Gates. Presently, however, under teleoperation, STAIR can maintain its particle filter distribution over the person's pose:



And given a specified target, plan a global path to get there that avoids laser-hit obstacles (it's following that path with DWA that's presently a bit of a difficulty):



6 Acknowledgements

I'd like to acknowledge Morgan and others on the `ros-users` list for providing invaluable debugging help after late nights spent frustrated with ROS. I apologize to Andrew for watching class on TV, though I received my comeuppance every time I wanted to ask a question. Huge thanks are also due to him for teaching both 221 and 229 this fall; fantastic classes, both. And much thanks goes to Olga for always being one step ahead of me and providing encouragement even when STAIR was walking away from the LED right into the wall.

References

[1] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4, 1997.

[2] Michael Montemerlo, Sebastian Thrun, and William Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *in IEEE International Conference on Robotics and Automation (ICRA)*, pages 695–701, 2002.

[3] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *International Conference on Robotics and Automation*, Open-Source Software workshop, 2009.

[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005.