Leg/People Detection Research Using Turtlebot Laser Scan Data (December 2016)

Amanda Loh and Chris Nakovski

***Abstract*— For the purpose of learning how to detect legs from a laser scanner on the Turtlebot. The TurtleBot uses the Kinect data to generate a laser scan that is published on the /scan topic. These laser scans, in real time, are processed by a filtering algorithm to filter out what observations are considered legs based on certain hyper parameters. Then, the legs are passed to a particle filtering algorithm to be filtered even further in order to distinguish legs more accurately, and ultimately people.**

***Index Terms*—autonomous, filtering algorithms, laser scan, leg detection, particle filter, people detection, ROS, turtle-bot**

I. INTRODUCTION

THE goal of this project was to detect people from the point of view of a smaller robot (we use a TurtleBot). The algorithm developed focuses on a simple and easy to understand approach which doesn't sacrifice accuracy when making classifications. We use ROS (Robot Operating System) to stream Kinect laser scan data by way of ROS topics. After that, we do some pre-processing to identify potential legs in the raw laser scan data. Then the observed legs are compared to a set of previously seen legs to create a new list of legs. This comparison is done by estimating the position of the previously seen leg by a cloud of particles around the point where the leg was observed, and calculating leg positions and variances as modeled by normal distributions. We pair up previously seen leg positions to observed potential legs, mutually using each's probability density function with the other's position to assign a correlation likelihood score. The most likely leg correlation matches are then accepted if they are within a close enough distance from the previous time step. This updates the position of the leg. We then weight particles around the leg and resample from them based on the likelihood that they belong to that leg. Once we have these particles, we propagate them through a motion model and add Gaussian noise. Then finally, potential pairs of legs are analyzed together to create an averaged person point that defines where the robot thinks a person is located in its field of vision.

II. THE LASER SCAN TOPIC AND INITIAL FILTERING

The data used to make predictions about potential legs in a Turtlebot's view is gathered from its laser scan topic: /scan. When subscribing to the /scan topic from the Turtlebot, four pieces of data are crucial for leg analysis. The first is the distance (in meters) of each laser beam. The second and third are the min and max angles of the robot's vision. The fourth is the increment of the angle between laser beams relative to the min angle. Every tenth of a second, a ROS topic publisher sends out a frame of data from the laser scan at the current time. The frame includes the four pieces of laser data. The data gathered per frame can be seen in Figure 1. The points plotted in this frame are shown in θ, r coordinates with θ being the angle of the laser scan relative to the robot from its θ_{min} position and r being the length of that laser beam in meters.

From the raw data, potential legs have to be filtered out from these initial dots. The way filtering was completed was by going point by point from left to right through the plot, looking for a large enough difference in distance between adjacent laser scan beams to be considered the start of a leg. This minimum delta between laser scan distances is a tunable parameter used for leg detection. Once a large enough distance delta is seen, the algorithm begins analysis of a potential leg and saves the start position of that leg. If the algorithm sees a rise in laser beam distance after the threshold drop that is greater in magnitude than our distance parameter threshold, the algorithm determines this is the end of a potential leg. The algorithm then takes the leg starting point and the point where the algorithm determines the end of the leg and takes the Euclidean distance between the two. The points are converted

to x,y from θ, r in order to get a distance in meters. This distance is then used to compare against a leg width min and leg width max value to determine if the width is within the bounds to be considered a leg. If it is, then the average of the two points are taken to determine the coordinate of the potential leg. With this algorithm, a plot can be made to determine initial filtering of legs. The algorithm provides a relatively good way to filter out legs with a few false positives scattered here and there. See Figure 2.

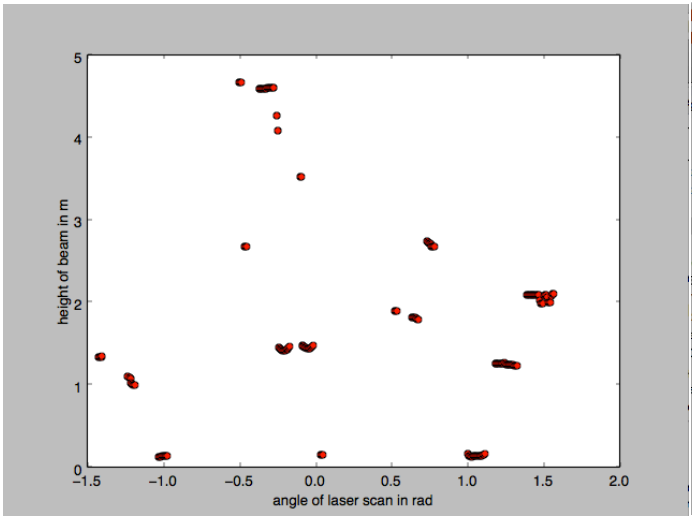

Figure 1. This is a plot of the raw laser scan data from the /scan topic in ROS. Notice the two semicircle horseshoe red dot areas, convex towards the x-axis, are the potential legs that are trying to be detected.

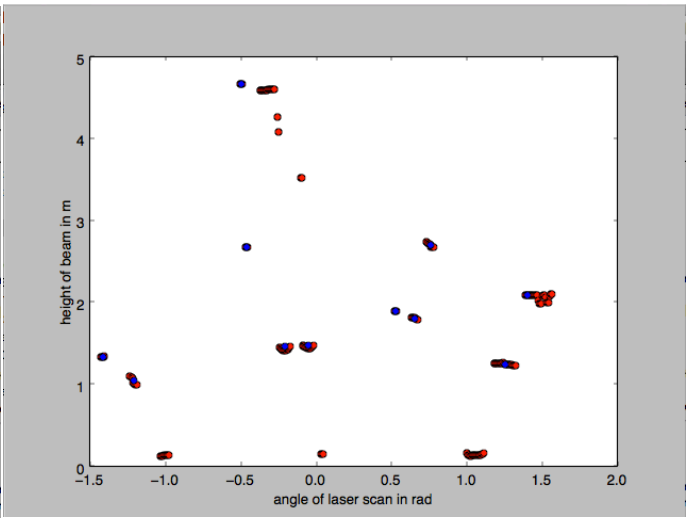

Figure 2. The same frame plot as in Figure 1, but with initial filtering implemented. Notice blue dots show where the algorithm thinks legs are. The two real legs are tagged, but there are also a lot of false positives.

III. Tuning Parameters for Initial Filtering

With the initial filtering algorithm, comes three parameters that can be tuned. The first is the drop delta which is how much of a drop in meters is needed from one point to another to determine a potential leg. The second is the min leg width: this is the minimum width of a leg to be considered in meters. The third is the max leg width: this is the maximum width of a leg to be considered in meters. Based on tuning experiments, it was found a drop delta of 0.1m and a tighter bound on leg widths created a more selective filter. With looser bounds, there were more false positives, but the algorithm got more classifications correct. Often times the tighter bound leg detector tuning would cause the simulations to miss a leg more often while moving through the laser scan frames. Figure 3 shows a tighter bound tuning and Figure 4 shows a looser bound tuning. Red points are the raw laser scan data and blue points are detected legs.

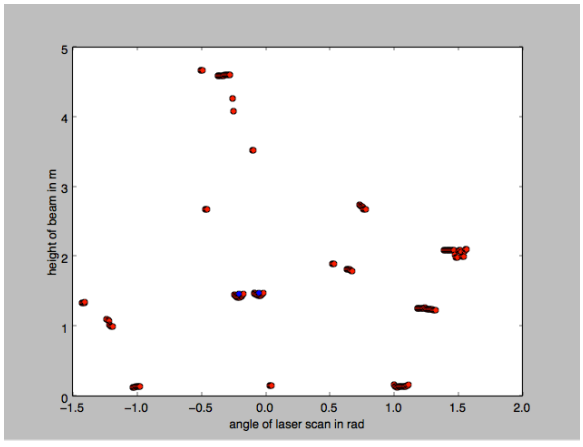

Figure 3. Tuning with a tighter bound. Although, in this frame, there seems to be high accuracy for leg detection, in real time, both legs are often not detected because leg widths are so tight. Drop delta is 0.1m, min leg width is 0.1m and max leg width is 0.12m

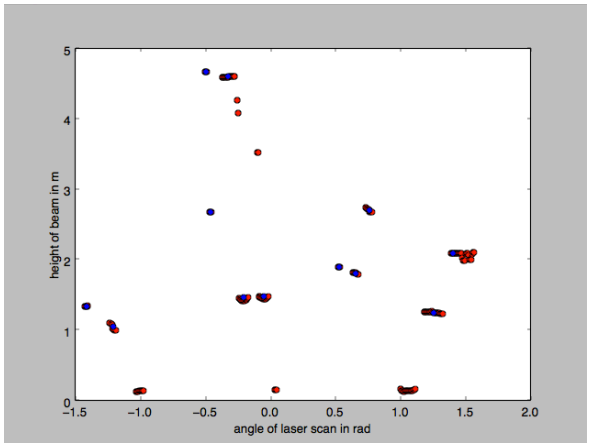

Figure 4. Tuning with a looser bound. Drop delta is 0.1m, min leg width at 0.001m and max leg width at 0.5m

Since a particle filter was also part of the leg detector algorithm, we decided to go with looser bounds for individual leg detection, as false positives for legs could later be filtered out when we compare legs to find people. In general, it's better not to discard a lot of data early on in your process, which is why we had looser bounds for detecting legs. In the end, the parameters we chose were in between the two thresholds in Figure 3 and Figure 4. The leg width min was 0.09m, the leg width max at 0.3 m, and the drop delta at 0.1m for the most successful accuracy of the algorithm.

IV. Checking further the validity of legs

After potential observed legs have been found, these legs have to compared to what legs have been previously seen before. For the very first observation frame, all observed legs become seen legs. However, in the case there have been legs seen before, each seen leg is compared to each observed leg. During the comparison, the correlation between the observed and seen leg will be determined by a score calculated from the product of each leg's pdf when compared to the other's mean. The highest pdf value between the seen leg and the observed leg will be saved. Then the Euclidean distance is taken between the well correlated seen and observed legs to make sure the points are close enough to be considered the same leg. If they are, then the observed leg is taken out of the observed list and added to the seen list of legs and a particle filter iteration is done on this leg given its previous seen update. If the observed leg does not match any seen legs, then it is added as a new leg with a brand new particle filter representation and a default variance of 30cm for both its x and y positions. This variance is updated for each iteration of the particle filter. Figure 5 shows a diagram of how this algorithm works.

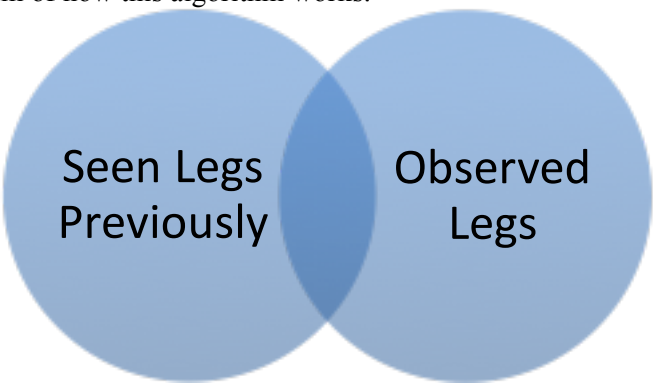

Figure 5. The dark blue area between the legs that have been seen and the legs that are currently being observed are the leg measurements that are filtered using a previous particle filter update. Points of legs seen already, but not observed, will be thrown out in this iteration/frame. Legs observed but never seen before will be added as a new leg for a new particle filter iteration.

V. Particle filtering

The particle filter gets passed an x and y point to the filter. Then, the dx and dy are either updated if the leg existed before based on position change, or initialized to zero the first time a leg is added. Upon leg initialization, 50 particles are sampled from a Gaussian around the observed leg's position with a variance of 30cm. Otherwise, the particle filter will reuse particles from a previous iteration and use them to calculate the new position, variance, and velocity of the leg. Below is the pseudo code for the particle filter:

Loop over particles:
- We take our current measurement
- Assign weights based on how close they are (Gaussian probability density)
- Resample based on weights
- Update position and variance based on new particles
- Propagate the particles through the motion model based on leg: dx, dy, and dt
- *we define dx to be currX-prevX, dy = currY – prevY

We convert our x and y point to a θ, r representation with the formulas:

$$r = \sqrt{dx^2 + dy^2}$$

$$\arctan\left(\frac{dy}{dx}\right) = \theta$$

Then we define a model with noise we want to add to our motion model which stems from past θ, r values from time t-1, plus a Gaussian noise:

$\theta_{withnoise} = \theta_{t-1} + gaussiansample(-\epsilon, \varepsilon)$

$r_{withnoise} = r_{t-1} + gaussiansample(-\epsilon, \varepsilon)$

We then propagate each individual particle through this motion model and use these new particles in the next iteration of our particle filter. See Figure 6 for the result of a particle filter iteration.

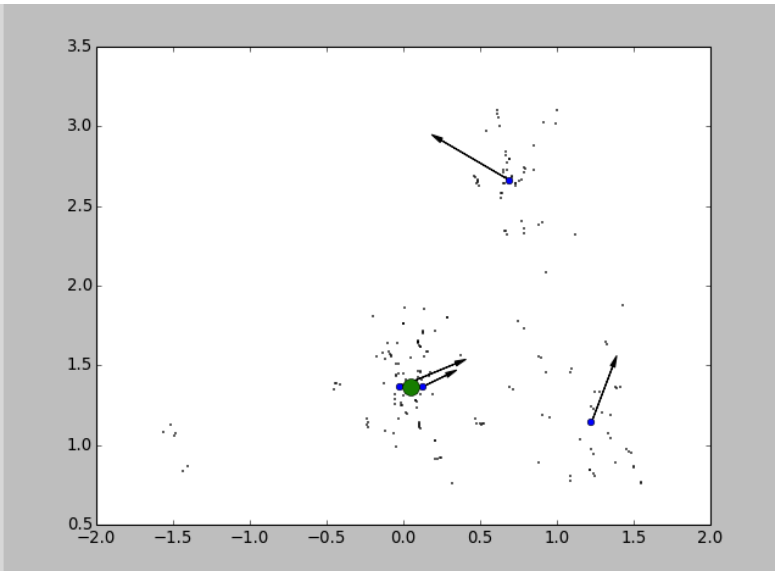

Figure 6. Successful classification of laser data samples of a person. In this live stream, there was one person in the frame. Added prediction of motion arrows were added as well to the plot. Blue points are legs, green dots are people, and small black dots are the particles from the particle filter

VI. Results

After tuning of the particle filter's distance threshold between legs, as well as the three parameters from the initial filtering, the algorithm was able to detect people effectively. Unfortunately, leg detection generated far more false positives than was hoped, but by giving the algorithm more data to work with, the people detector in the particle filter was able to accurately determine a person. If more legs, were filtered out, through tuning, the algorithm would be unable to detect people. This algorithm was tested on two bag files: both of which contained a person walking around the Turtlebot. The second bag file contained objects such as chairs, desks, and so on that easily appeared in laser scans to be legs that could be characterized as a person. However, with the motion model of the particle filter used in conjunction with the initial filtering, these obstacles were not detected as people. The algorithm was also tested with two people as well to show the laser scan could detect more than one person at a time. See Figure 7.

One of the issues that still could be improved about the algorithm is that sometimes people are not detected at certain frames due to unusual leg scan data configuration. These frames usually occur when the person is facing the TurtleBot at an awkward angle where the robot cannot see the distinct semi-circle shapes of the legs the algorithm uses to characterize legs. Perhaps this issue could be resolved with additional scan data from other sources at different angles and an alternate prediction algorithm. Despite, these frames where the robot doesn't detect the person, the next frame usually catches the person at a better angle and the green dot for the person displays again.

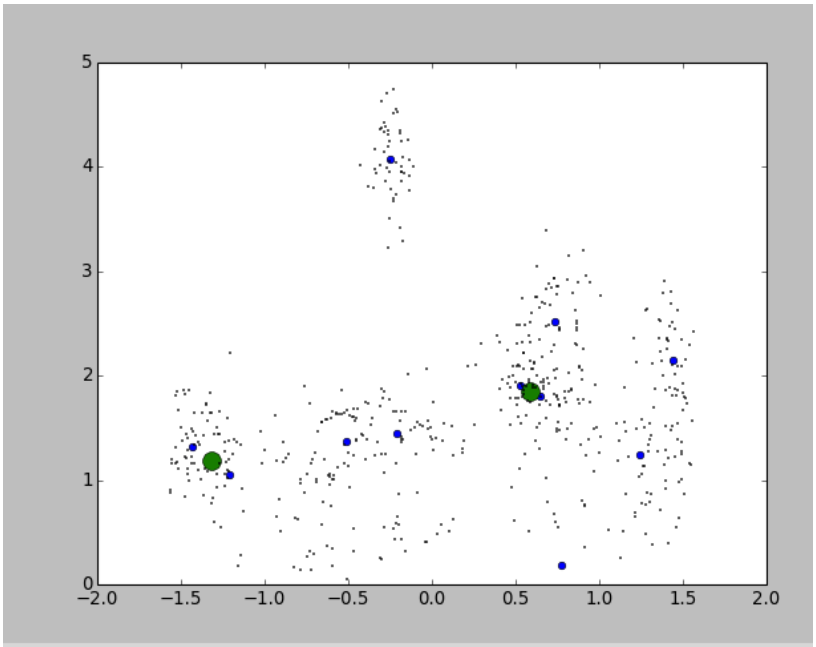

Figure 7. Two-person particle filtering with the y-axis distance of the scan in meters and the x-axis being the angle of the scan.

VII. Conclusion

Overall, this project was a success. However, getting to this success was extremely challenging. As two undergraduate students who did not know ROS well and had not implemented a particle filter on a real robot with noise, the project proved to be a difficult challenge. After, writing a lot of pseudo code and spending a lot of time planning out what was to be done, the algorithm produced was effective in achieving the goal of detecting legs and people.

An improvement that could have been made is that when a person disappears off the graph, there can be an additional prediction made based on where the person used to be. This prediction can verify that the person is most likely still close by and the person still can be tracked even though there is an occlusion due to the observation data where the legs cannot be characterized clearly. Also, multi-threading could have been implemented to make particle propagation/graphing faster and to be able to increase the number of particles for the particle filter for more accuracy in leg and people detection.

References

[1] J. Huang, "How to run the leg detector on Turtlebot", *GitHub*, 2014. [Online]. Available: https://github.com/jstnhuang/cse481c_tutorials/wiki/How-to-run-the-leg-detector-on-the-Turtlebot. [Accessed: 11- Dec- 2016].

[2] D. Lu, "leg_detector - ROS Wiki", *Wiki.ros.org*, 2014. [Online]. Available: http://wiki.ros.org/leg_detector. [Accessed: 11- Dec- 2016].

[3] "ROS/Tutorials - ROS Wiki", *Wiki.ros.org*, 2016. [Online]. Available: http://wiki.ros.org/ROS/Tutorials. [Accessed: 11- Dec- 2016].

[4] "matplotlib: python plotting: Matplotlib 1.5.3 documentation", *Matplotlib.org*, 2014. [Online]. Available: http://matplotlib.org. [Accessed: 11- Dec- 2016].

[5] Andreas Svensson. (2013, Oct 8). Particle Filter Explained without Equations. [YouTube video]. Available: https://www.youtube.com/watch?v=aUkBa1zMKv4&t=145s. Accessed Dec. 11, 2016.