Rachel Boy, Kyle McConnaughay, Brooks Willis
Computation Robotics - Fall 2014
October 16, 2014

Laser-Corrected Odometry

Mobile Robotics Project Write-up

Figure 1: #mission #accomplished

Project Goal

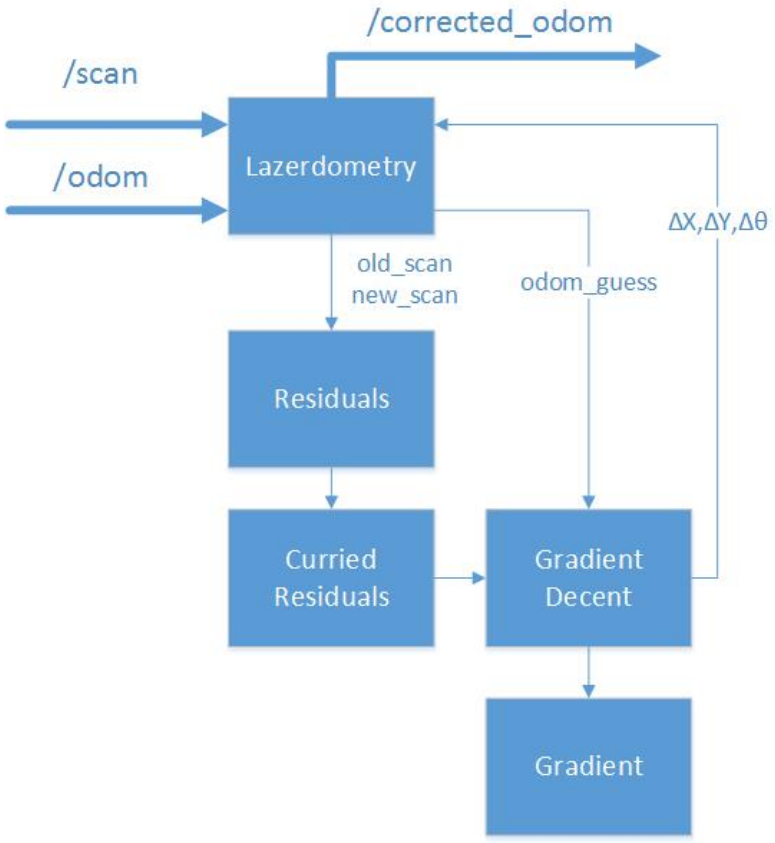
We set out to augment odometry with laser scan data to more accurately predict the movement of our robot. Odometry has several limitations that laser scan data could be used to correct. For instance, a robot that collides with a wall or slips on a slippery surface will have inaccurate odometry. The laser scan data, however, could be used to recognize the fact that the robot's actual movement does not correspond with what the odometry reports.

Solution Description

We use gradient descent to guess values for how far the robot has moved/turned in order to minimize the measured difference between consecutive laser scans. Given three parameters, change in x, y, and theta, we compute a transform of the most recent scan to the reference frame of the scan before it. We then compute the residual between the two scans. Our residuals function is defined by the squared error between the points of two successive laser scans. Gradient descent was used to determine which values of delta x, delta y, and delta theta minimized the residuals function across each pair of successive scans. Odometry provided the initial guess for delta x, delta y, and delta theta that was fed into the gradient descent. These values were then output to a ROS topic for comparison to the raw odometry measurements.

To show that our solution was successful, we were going to tele-operate the robot on a semi-random path and then return it to the starting position and orientation. This would allow us to easily assess what the accrued error was over the course of travel for both the unmodified odometry and our corrected odometry.

System Architecture



Our code is a mix of object oriented and functional. “Lazerdometry” is a black box that contains three classes, “Lazer”, “Odom”,and “Lazerdom”. “Lazer” handles the input from the lidar, and processes it into a form that can be passed down to “Lazerdom”. “Odom” does the same for the odometry. “Lazerdom” is where the high level logic takes place. Using the data from the other two classes, it determines how far the robot has moved since its last update. To do this, it invokes the functional part of the code. The old and new scan data is transformed and passed to the curried residuals function. This curried residuals function is passed to gradient descent, which finds its local minimum, using the information from the odometry as a starting guess. The dx, dy, and dtheta which give a local minima are passed back up to “Lazerdom” as an approximation of the robots movement, and used to inform the next round of computation.

The object oriented versus functional breakdown occurred because the processes in Lazerdometry require some degree of memory from scan to scan so that there is some baseline for the calculations to work from. The functions do not then need this memory built in since they work on only a pair of time steps at a time.

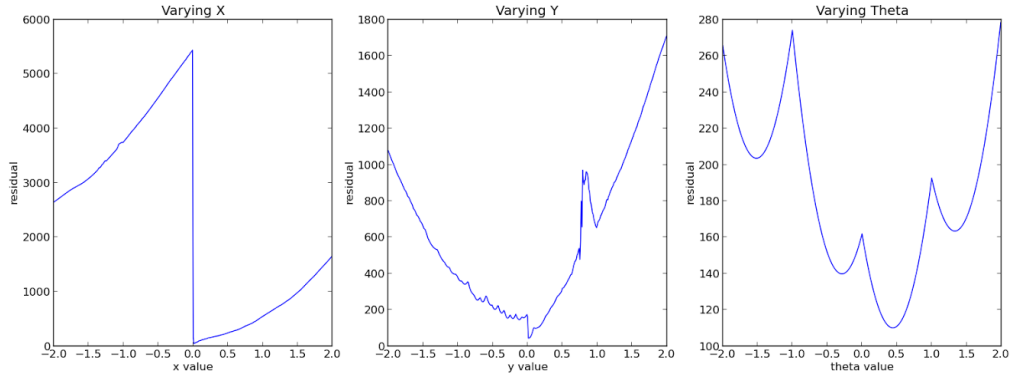
Design Decisions

The largest design decision we made was choosing a method for determining which values of delta x, delta y, and delta theta best described the changes in laser scan data. The two ideas we considered were gradient descent and a bayesian particle filter. Because gradient descent was a simpler, less time-consuming approach, we decided to forgo the particle filter. The gradient descent approach appeared as if it would yield a fairly fast algorithm which was fairly accurate, since the math involved is comparatively simple and it is only running on a 360 point data set at any one time.

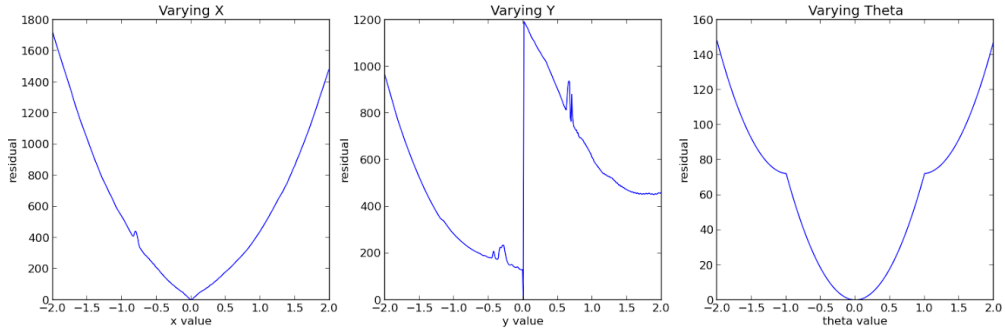
Challenges Faced

Gradient descent was a poor choice. The analytic approach was only as accurate as the error tolerance that we specified. Gradient descent consistently predicted a slow, diagonal drift in the robot’s location on the same order of magnitude as the absolute error we specified for convergence in gradient descent. Additionally, the residuals function was not well-behaved (as shown in the figures below); there are numerous places where local minima do not coincide with absolute minima, which also increases the error at each step. Unfortunately, we discovered this problem too late in the project to implement a different solver. Which leads us to future steps!

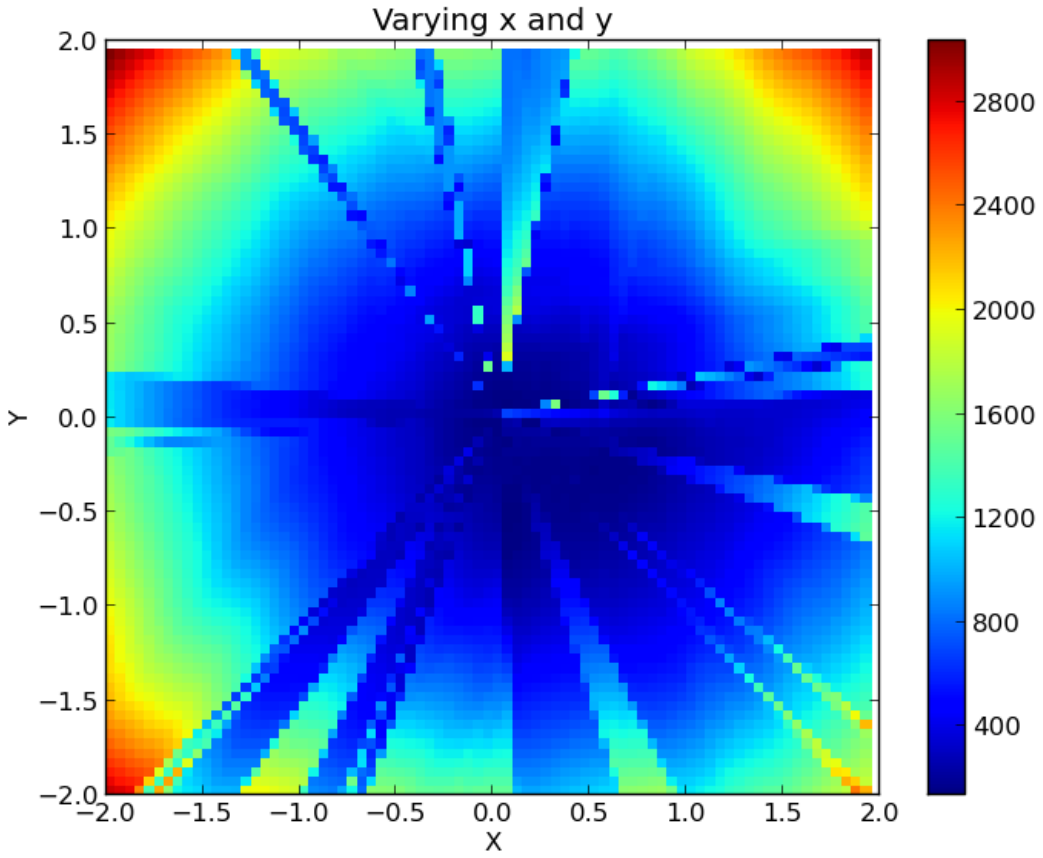
A



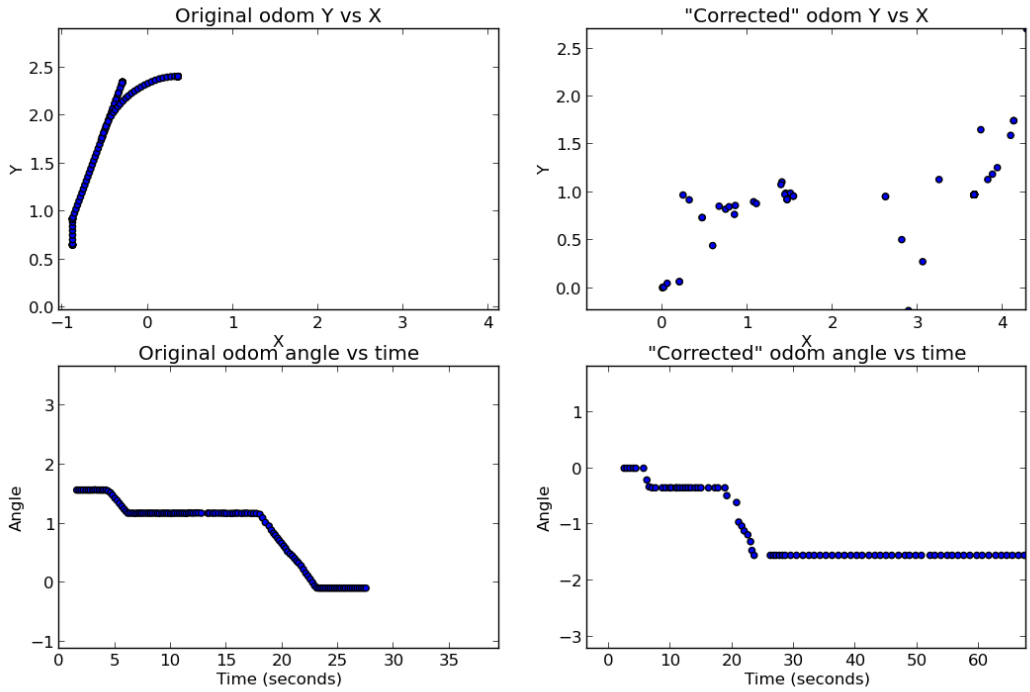
B



The plots above are slices of the residual function. Rows A and B represent two different points in time. For each point in time, we vary our hypothesized translation along the x axis, holding y and theta constant, then do the same for translation along the y axis and rotation in theta. (For all of these, 0 on the independent axis represents the guess from the odometry). The discontinuity the residuals function varying along x (for sample A) and y (for sample B) is, at the time of writing, totally unexplained. But it is clear that not everything is roses and sunshine with these functions. (The poor behavior when varying along theta is more easily explained, as we only have data points every one degree, and are interpolating between them, and the function gets wacky every degree or so.)



This plot shows the effect on the residual of varying both dx and dy (leaving dtheta constant). We believe the appearance of rays comes from the range of the LIDAR - in areas that were far enough away that their apparent distance did not change between frames, we observe one residuals function, and for objects we could see, we observe another. The local minima for both does seem to be near the guess made by the odometry, however (centered at (0,0)), indicating that our solution has some theoretical justification.



The plots in the left column show the odometry reported by the robot. The plots in the right column show the odometry reported by our system. They are not the same, which in this case is actually correct since the robot collides with a wall about 1 meter away from the starting spot. You can see in the original odometry where we run into a wall (the odometry believes we keep going forward - we back up to get away from the wall). The “corrected” odometry identifies the fact that the robot does not move while running into the wall. Unfortunately, due to the poorly behaved residuals and the computational latency, its measurements are not very accurate. One note about the plots; the upper left plot has the robot starting at (-1, 0.5) while the upper right plot has it starting at (0, 0). This is because we started the lazerdometry after we had already driven the robot via teleop.

Future Steps

We believe that implementing a bayesian filter (rather than gradient descent) would significantly reduce the error produced by the current method. Exchanging gradient descent for the particle filter would be our first step to improve this project. That swap could be made without changing other parts of the program considerably, since the two methods use very similar data and output results in the same format.

Other areas of interest include improving the residuals function (simply using smarter, less brute force techniques), handling noise in the data more robustly, and doing more rigorous analysis of the improvements (or lack thereof) afforded by gradient descent and the particle filter versus raw odometry. Each of these improvements could stand alone or in conjunction fairly easily.

Lessons Learned

We believe that the reason probabilistic approaches to robotics problems are popular is because they are more robust against noisy environments and can be optimized for speed more easily that analytic

approaches. There are methods for improving the performance of gradient descent on poorly behaved manifolds, like adding random noise to each step, but these methods also tend to increase the run time of gradient descent, which would have exacerbated our already considerable performance problems. A bayesian approach probably would have been a better choice.

Future projects would also benefit from integrating code into a functioning ROS node earlier. We would have discovered issues with our transforms much earlier if we had done continuous integration. We used unit tests on all of our code, however we underestimated the complexity of the tests required to fully flush out the errors in a number of areas, which led us calling code done and tested before that was in fact the case. Future projects will definitely contain far more thoroughly thought out tests. Additionally, the earlier ROS integration will allow Bag files to be fed in sooner, making testing easier to perform on real data.