RangeTrack: Object Detection, Object Identification, and Robot Localization
with the Sharp "ET" Rangefinder (Part 2)
Jeremy Rand and Emily Curtis
Norman Advanced Robotics
jeremy.rand@ou.edu, coconut231@yahoo.com

RangeTrack: Object Detection, Object Identification, and Robot Localization with the Sharp "ET" Rangefinder (Part 2)

8 Welcome to Part 2!

Welcome back to RangeTrack. In Part 2, we'll cover what you can do now that you have the filtered data obtained in Part 1.

9 Geometry Projection

After noise reduction is performed, we have a set of sensor locations and a set of range distances. To find where the actual detected object is, we use trigonometry to project the location of the point on the object which the rangefinder detected. The equations are:

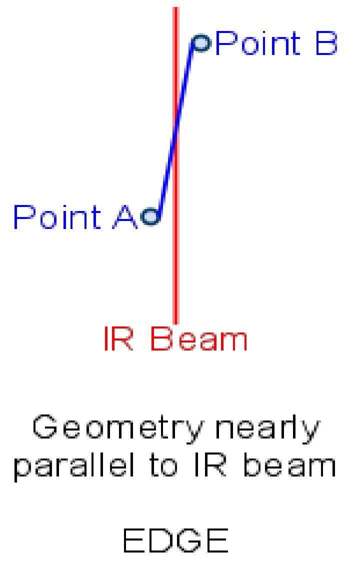
$$(x_0, y_0) = Sensor\ Position$$
$$\theta = Sensor\ Orientation$$
$$r = Range$$
$$(x, y) = Detected\ Point$$

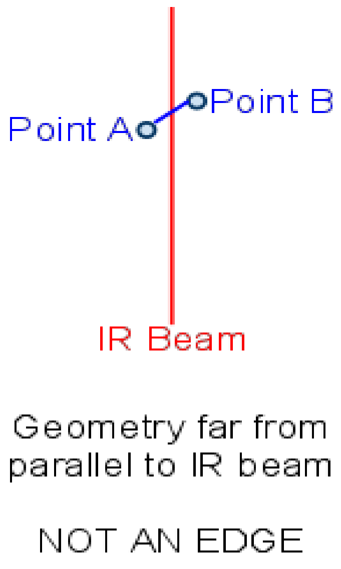
$$x = x_0 + r \bullet cos(\theta)$$
$$y = y_0 + r \bullet sin(\theta)$$

As with the ET measurement, we used lookup tables for the sin and cos functions for better speed.

10 Edge Detection

Geometry projection gives us the shape of the detected points; next, we have to divide the points into discrete objects. This is implemented by comparing the Theta orientation of the sensor with the angle of the tangent line at a specific point on the projected geometry. If the two angles are very close, then the geometry is nearly parallel to the sensor beam, indicating that the beam just entered or exited an object (and is therefore an edge).



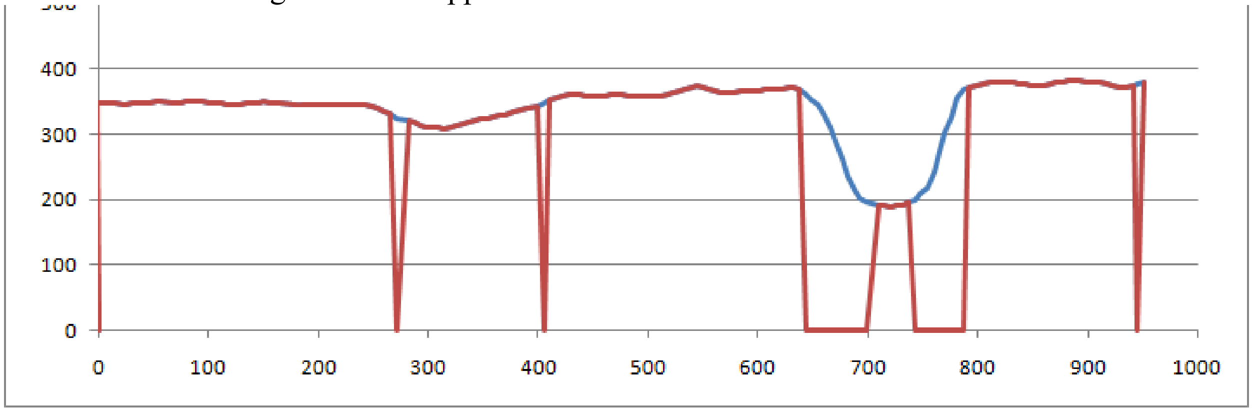


Edge detection is the process which is most sensitive to the noise reduction settings; if noise is too intense, false edges will appear, while if the data is smoothed too much, legitimate edges may be ignored.

The edge detector takes a few inputs. A minimum and maximum range can be provided, since very far away objects are unlikely to be what you're looking for, and the limited resolution at such long range will probably result in more noise and more false edges. The edge detection threshold, which specifies what slopes are strong enough to be considered edges. can also be specified. A maximum range of 500mm and an edge threshold of +/-150mrad of the line of sight worked well for us. But remember, you'll need to tune these settings for your own data.

One non-obvious bit of tuning that helps is to remember that the ET's beam is asymmetrical; if the ET is rolled horizontally relative to its beam, the left side of an object looks different from its right side. This may necessitate using edge thresholds which aren't a simple +/- value but which have different magnitudes depending on the sign. Alternatively, rolling the ET vertically relative to its beam will make its beam horizontally symmetrical, which may be more desirable and can make the RangeTrack tuning process simpler.

A visualization of edge detection applied to the data visualized above is below:



(Red Zeros Indicate Edges Detected)

11 Statistical Calculation

Once we know that there is a discrete object, we probably want to know what it looks like. This is done using statistical calculations. Statistical functions can be called at any time, and will work with the RangeTrack data from the most recent `UpdateAll()` call. Currently available statistics include:

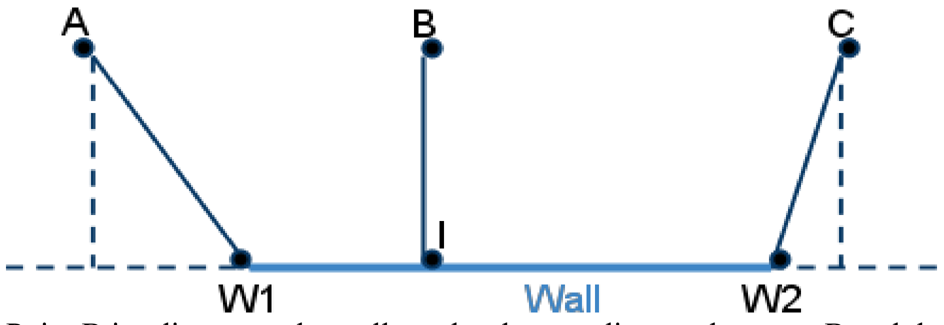
- Number of segments found (don't request data for segment ID's at or above this number; you will get a crash).
    - `rt->GetNumSegments();`
- Starting and ending positions of the sensor (odometry values)
    - `rt->GetSegmentStartPositionX(int segment);`
    - `rt->GetSegmentStartPositionY(int segment);`
    - `rt->GetSegmentStartPositionTheta(int segment);`
    - `rt->GetSegmentEndPositionX(int segment);`
    - `rt->GetSegmentEndPositionY(int segment);`
    - `rt->GetSegmentEndPositionTheta(int segment);`
- Starting and ending locations of the object
    - `rt->GetSegmentStartPointX(int segment);`
    - `rt->GetSegmentStartPointY(int segment);`
    - `rt->GetSegmentEndPointX(int segment);`
    - `rt->GetSegmentEndPointY(int segment);`
- Object length
    - `rt->GetSegmentLengthPointX(int segment);`
        - X component of distance from first point to last point of the object
    - `rt->GetSegmentLengthPointY(int segment);`
        - Y component of distance from first point to last point of the object
    - `rt->GetSegmentLengthPointChord(int segment);`
        - Entire distance from first point to last point of the object
- Minimum range
    - `rt->GetSegmentMinRange(int segment);`
- Maximum range
    - `rt->GetSegmentMaxRange(int segment);`
- Mean range
    - `rt->GetSegmentMeanRange(int segment);`
- Mean angle
    - `rt->GetSegmentMeanAngleSlope(int segment);`
- Not yet implemented:
    - Arc length
        - Distance from point to point for the entire object
    - Standard deviation of angle
    - Mean concavity
    - Standard deviation of concavity

A pom probably has a small length, a relatively variable angle, and a relatively high concavity, while a wall probably has a large length, a relatively stable angle, and a relatively low concavity.

12 XY Localization

The ability to distinguish walls from game pieces yields a very useful functionality: localization. Localization is the opposite of standard object detection: rather than knowing where the robot is and finding the location of game pieces, localization knows where the wall are and finds the location of the robot. Since we wanted to gain some practical education from this research, our localization algorithm is entirely homemade; we didn't consult any websites regarding how the algorithm should work. As a result, there may be better algorithms that seasoned robotics engineers know about, but our algorithm is quite effective.

The first step is to find the vector yielding the shortest distance between points obtained by geometry projection and the known wall layout of the game board. For each combination of a point and a wall, the distance is a piecewise function depending on where the point is relative to the wall's endpoints, as shown below:



Point B is adjacent to the wall, so the shortest distance between B and the wall is perpendicular to the line containing the wall. Points A and C fall to the side of the wall, so the shortest distance between them and the wall is the distance between A and the left wall endpoint, and C and the right wall endpoint. The mathematical method to detect this is measuring angles: $\angle AW_1W_2$ is obtuse, indicating that A lies beyond W_1. Similarly, $\angle W_1W_2C$ is obtuse, indicating that C lies beyond W_2. $\angle BW_1W_2$ and $\angle W_1W_2B$ are both acute, indicating that B is adjacent to the wall. The advantage of this method is that it will work for any wall and any point, regardless of location or orientation.

Actually calculating the angle based on the point coordinates can be done via the `atan2` function, which converts an XY point into the angle between the positive X axis and the line which intersects the specified point and the origin. For XY points A, B, and C, and origin O::

$$\begin{aligned}
Angle\ OBA &= atan[(A-B)_y,(A-B)_x] \\
Angle\ OBC &= atan[(C-B)_y,(C-B)_x] \\
Angle\ ABC &= Angle\ OBC - Angle\ OBA
\end{aligned}$$

Once we've calculated the angles, in the obtuse cases, the vector distance between the point and the end of the wall is yielded by subtracting their coordinates. For the both-acute case, finding the vector distance between the point and the middle of the wall is harder. We ended up using the formula from Will Garner [7], which related the point of intersection (I on the above diagram) to the slope *m* and y-intercept *b* of the wall:

$$I_x = \frac{mB_y + B_x - mb}{m^2+1}$$
$$I_y = \frac{m^2B_y + mB_x + b}{m^2+1}$$

m can be computed based on W_2 and W_1:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

b can be computed by solving the slope-intercept form of a line for *b*:

$$y = mx + b$$
$$b = y - mx$$

These equations give us a vector distance between the point and the wall, for any case. Taking this calculation for all walls and choosing the vector with minimum magnitude yields the shortest vector distance between the point and the entire set of walls.

We then run this calculation for all points, and take the mean. The resulting vector is the mean difference between the known wall location and the observed wall location. Just take the XY coordinates where you think your robot is (as measured by odometry), add this vector to them, and you now have the corrected robot location!

13 Angle Localization

The XY localization algorithm above will correct XY coordinates, but is unable to correct the robot's orientation. Our solution is simply to repeat the XY localization algorithm after rotating the wall layout by a few degrees, and measure how well the localized XY position fits the original geometry projection data. Rotation of walls' endpoint is calculated by recalculating the XY localization using the point *(x', y')* for each endpoint instead of the usual *(x, y)*, as per the following 2D rotation formulas [9]:

$$x' = x\ cos(\theta) - y\ cos(\theta)$$
$$y' = x\ sin(\theta) + y\ cos(\theta)$$

Measuring how well the rotated walls fit our data is done by taking the set of vector difference between the known wall location and the observed wall location (the same set of data we previously found, whose mean is the XY correction), and calculating the Mean Absolute Deviation (M.A.D.) of the data instead of the mean. The M.A.D. is a simple statistical measure of the spread of a data set; it is defined as the mean distance between each data point and the mean of the data set. Higher M.A.D. values indicate highly spread out values, which in our case would indicate that the rotated walls don't fit the observed walls. Conversely, lower M.A.D. values indicate that the rotated walls are a good fit for the data.

The wall layout with the lowest M.A.D. best fits the data, and the rotation of that wall layout is the correction which should be added to the orientation measured by odometry.

Using X/Y/Angle Localization works like this:

```
double old_rotation = 0.0, old_x = 0.0, old_y = 0.0; // We think we're
parallel to the X axis, at the origin

double rotation_step = 5.0 * 3.14159 / 180.0, rotation_range = 20.0 * 3.14159
/ 180.0; // Try adjusting the rotation by increments of 5 degrees, with a
maximum change of +/- 20 degrees

double new_rotation, result_mean_x, result_mean_y, result_mad_x, result
mad_y;

double new_x, new_y;

rt->AddSegmentAsWall(0); // The first and third objects seen are walls.
rt->AddSegmentAsWall(2); // (Use statistical calculations to decide.)

rt->ComputeLocalizationStatsWithAngle(old_rotation, rotation_step,
rotation_range, new_rotation, result_mean_x, result_mean_y, result_mad_x,
result mad_y);

new_x = rt->RotatePointX(old_x, old_y, new_rotation-old_rotation) +
result_mean_x;
new_y = rt->RotatePointY(old_x, old_y, new_rotation-old_rotation) +
result_mean_y;
```

XY and angle localization do *not* use lookup tables; rather, they use double-precision floating-point math, and are therefore slower. However, since they usually only need to be called occasionally, this is not a huge problem (although we may investigate lookup tables for future versions). The exact speed is dependent on the number of data points, number of walls, and number of angles to check, but as an example, a data set of 174 points with 1 wall and 16 angles tested took 922ms. At the moment, we're still adding the code to allow the user to intelligently specify which objects are walls; right now it treats all points as part of a wall (we expect to have this finished before GCER).

14 CSV Dumping

Manually testing filtering parameters and statistical thresholds on a physical robot can be a large time sink. To minimize this bother, RangeTrack can dump all of its data into CSV files, which Excel can import for analysis. This functionality can be called via the `rt->DumpLog(filename)` and `rt->DumpSegment(filename)` functions. To dump to a USB drive, use filenames in the path `/mnt/browser/usb/` . To dump via WiFi, use filenames in `/tmp/` and simply use the following command from a Bash prompt on the PC to copy the files:

```
scp root@CBCIP:/tmp/*.csv ./
```

RangeTrack includes an Excel workbook which can simulate most of RangeTrack's CBC-side features, including filtering and edge detection, complete with pretty graphs. This is highly useful; just dump a scan that you want to process, copy the data into Excel, and play with the parameters until you find reliable parameters. Anyone with basic familiarity with Excel should be able to understand what's going on, but a detailed guide for the Excel simulation will be posted on the Botball Community prior to GCER.

15 Resetting RangeTrack

At times, you may want to reset the data which RangeTrack has gathered without recreating all of the C++ objects. To do this, call `rt->Reset();` . This will delete all of the data produced by logging, noise reduction, geometry projection, and edge detection.

16 Future Work

We're interested in merging data from multiple rangefinders for localization, as well as accepting data from non-rangefinder sources to be merged for localization. For example, a Top Hat sensor could generate XY coordinates where it saw a black line, and the walls could be the known location where the black line is. RangeTrack is designed so that this would be relatively easy. The Constant odometry sensor could easily be rigged to provide the location of the Top Hat, and a Range sensor could provide 0mm if the line is detected, and 100mm if the line is not detected. A maximum range of 50mm would then result in objects being logged whenever the Top Hat passes over a black line. XY localization data could be retrieved as usual and the multiple sensors' data simply averaged (with weight, if desired). The only part of the core code that would need work would be the angle localization so that it can work with multiple sets of XY localization data.

17 Conclusion

RangeTrack offers a very nice alternative to the CBC's camera, and can successfully operate in situations which would cripple the camera's functionality. We'd like to thank Mr. Tracy Foor of Norman North High School and Dr. Curtis McKnight of University of Oklahoma for being excellent statistics teachers; without the knowledge gained from their statistics classes, RangeTrack wouldn't exist. (Mr. Foor and Dr. McKnight were not significantly involved in this project specifically.) If you have questions, comments, or suggestions for RangeTrack, we can be found at the Botball Community [5]. Happy Tracking!

References

[1] J. Rand, M. Thompson, B. McDorman. Hacking the CBC Botball Controller: Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked. Proceedings of the 2009 Global Conference on Educational Robotics, July 2009.
[2] J. Rand. CBC Hacking 2011: Vision Enhancements and Sensor Speedups. Proceedings of the 2011 Global Conference on Educational Robotics, July 2011.
[3] J. Rand, M. Thompson, B. McDorman. CBC Hacking 2010. Proceedings of the 2010 Global Conference on Educational Robotics, July 2010.
[4] J. Rand. JeremyRand/RangeTrack - Github. https://github.com/JeremyRand/RangeTrack/ , June 2011.
[5] Botball Youth Advisory Council. Botball Community. http://community.botball.org , June 2011.
[6] J. Rand, M. Thompson. NHS Patchset. https://github.com/JeremyRand/cbc , June 2011.

[7] W. Garner. Shortest Distance from a Point to a Line. http://math.ucsd.edu/~wgarner/math4c/derivations/distance/distptline.htm , June 2005.
[8] G. Reshko, I. Nourbakhsh, M. Mason, G. Zeglin, J. Mouch, B. Velentini, A. Demke. Palm Pilot Robot Kit: Technical Information. http://www.cs.cmu.edu/~pprk/tech_info.html , October 2000.
[9] Wikipedia contributors. Rotation (mathematics). http://en.wikipedia.org/wiki/Rotation_%28mathematics%29 , May 2011.