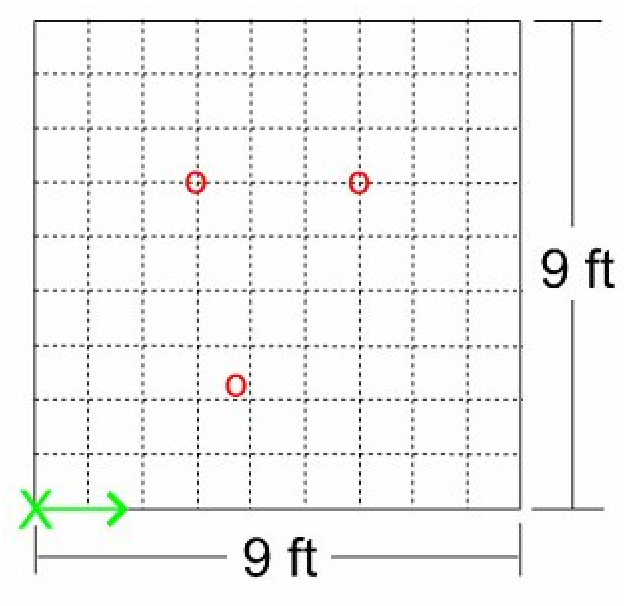
Final Project – SLAM with ER1 Robot

Part I: Data Acquisition

The objective of this final project is to acquire real odometry and laser scan data on your own using the Evolution Robotics platform (ER1). Using this data, Matlab code from homework 4 can be leveraged to run a SLAM simulation offline. To simplify things, set up a square grid for the ER1 to explore. In room 279, the tiles are 9 inches by 9 inches so a 9 foot by 9 foot square grid is convenient. The red circles in the figure below are obstacles (i.e. landmarks).



Download the Visual Basic code from the SLAM web site onto the ER1 laptop. This code is setup to do the following:

- Move the ER1 forward 9 inches
- When move has completed, get position data from encoders
- Execute a single scan of environment
- Write encoder and laser data to a text file
- loop again

So all you have to do is place the ER1 in its start position (pick any corner of the square) and have it face the next corner assuming a counterclockwise movement around the grid. Note that it is important to have the ER1's longitudinal axis exactly even with the grid border. Ten degrees of angular error can be detrimental to SLAM (assuming nearest neighbor data association). Ideally, the ER1 should return to the exact position it started from after running the code. However, accumulation in encoder error will prevent this. Be sure to keep an eye on the robot's true position (either visually or with an overhead camera) to compare to your SLAM results.

Part II: SLAM Simulation

The first thing that is needed is to formulate a model of the ER1 platform. Typically, a simple vehicle model is given by:

$$\begin{bmatrix} \dot{x}_v \\ \dot{y}_v \\ \dot{\phi}_v \end{bmatrix} = \begin{bmatrix} V \cos \phi_v \\ V \sin \phi_v \\ V \tan \alpha / L \end{bmatrix}$$

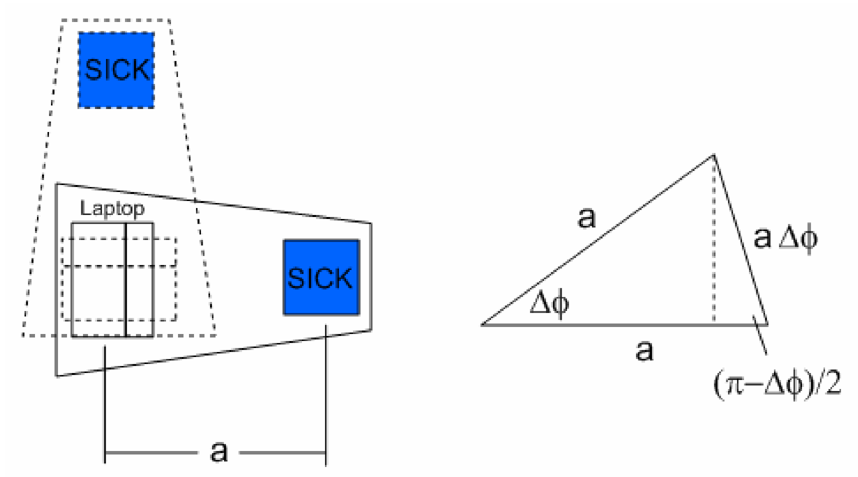
where the v subscript denotes the vehicle position. Putting this in the form of a difference equation, we have

$$\begin{bmatrix} x_v(k+1) \\ y_v(k+1) \\ \phi_v(k+1) \end{bmatrix} = \begin{bmatrix} x_v(k) + \Delta t V \cos \phi_v \\ y_v(k) + \Delta t V \sin \phi_v \\ \phi_v(k) + \Delta t V \tan \alpha / L \end{bmatrix}$$

However, since this robot is not in constant motion and we are getting x,y,ϕ coordinates from the encoders, it's more appropriate to formulate the model in this form:

$$\begin{bmatrix} x_v(k+1) \\ y_v(k+1) \\ \phi_v(k+1) \end{bmatrix} = \begin{bmatrix} x_v(k) + \Delta x_{trans+rot} \\ y_v(k) + \Delta y_{trans+rot} \\ \phi_v(k) + \Delta \phi \end{bmatrix}$$

This model is still slightly inaccurate. We'd like to transform everything into the laser scanner frame. This is slightly complicated because it only really comes into play during a rotation. The ER1 rotates in place and will therefore show no change in the x and y position when read from the encoders. However, since the SICK laser scanner is at some distance "a" from the ER1 laptop, it will move to a new x,y location. This is shown in the picture below:



It can be seen from the picture on the right that as the robot turns $\Delta\phi$, the change in the x and y values due to rotation are

$$\Delta x_{rotation} = a\Delta\phi \cos\left(\frac{\pi - \Delta\phi}{2}\right) \qquad \Delta y_{rotation} = a\Delta\phi \sin\left(\frac{\pi - \Delta\phi}{2}\right)$$

However, you also have to take into account the robot's current orientation. The final process model of the ER1 becomes:

$$\begin{bmatrix} x_L(k+1) \\ y_L(k+1) \\ \phi_L(k+1) \end{bmatrix} = \begin{bmatrix} x_L(k) + \Delta x_{translation} + a\Delta\phi \cos\left(\phi + \pi/2 + (\pi - \Delta\phi)/2\right) \\ y_L(k) + \Delta y_{translation} + a\Delta\phi \sin\left(\phi + \pi/2 + (\pi - \Delta\phi)/2\right) \\ \phi_v(k) + \Delta\phi \end{bmatrix}$$

where the L subscript denotes the position of the laser scanner. From here, you should be able to leverage your homework from last week. To simplify things, use only the laser scanner data:

- that forms the left sector of the scan (i.e. from 90 to 180 degrees). Since the ER1 is navigating counterclockwise around the perimeter of a square, it should only be looking to its left for obstacles within the grid.
- that does not include other obstacles (i.e. tables, chairs, etc.). Tthis should be about half of your data set.

Obviously, this is not ideal but it will definitely make things less complex.