RAFS - Robot Aided Feng Shui

Coding/Algorithm Design Document

Peter Motykowski-|-Bradley White-|-J.D. Pohlman-|-Matt Allen

Table of Contents

Wander Module

Assumptions:

- The laser and camera have powered on.
- The robot must be in its predefined starting position for localization.

Algorithm Description:

The wander module is fairly simple to follow. It calculates a random X, Y position in the room. The psuedo formula is:

```
if(rand() % 2 ==0)
        X=(((rand()*rand()) % (lower x_bound))*-1)
else
        X=((rand()*rand()) % (maximum x_bound))
if(rand() % 2 ==0)
        Y=(((rand()*rand()) % (lower x_bound))*-1)
else
        Y=((rand()*rand()) % (maximum x_bound))
```

These formulas calculate a global X, Y position for the EB 2029 based on our laser map. The gradient then takes this X, Y making it a goal for the robot. If the goal X, Y is inside an object that is defined by our laser map, the gradient will fail to find a path and look for another goal. When seeking a goal, the wander module uses gradient path finding and laser localization to go from its current position to the next goal position. These Saphira classes give us dynamic obstacle avoidance needed to get from point A to point B.

Object Recognition Module

Assumptions:

- Chairs are marked with the appropriate color tag (the color the robot associates with a chair).
- The wander module has been loaded and invoked.

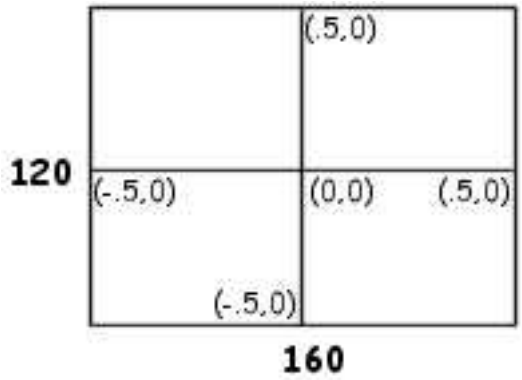
Algorithm Description:

This module employs two of the sensory devices mounted on the robot. First is the camera that is used for color recognition. We marked the chairs with pink paper and trained the ACTS software to respond to this color. Using the Saphira interface to ACTS, we queried the trained color channel for recognized objects (blobs). ACTS returns the location of blob data in terms of XY coordinates. These XY coordinates map to the 160x120 frame captured by the camera. In order to translate these coordinates to something meaning to the robot we had to use the following formula:

WIDTH = 160
HEIGHT = 120

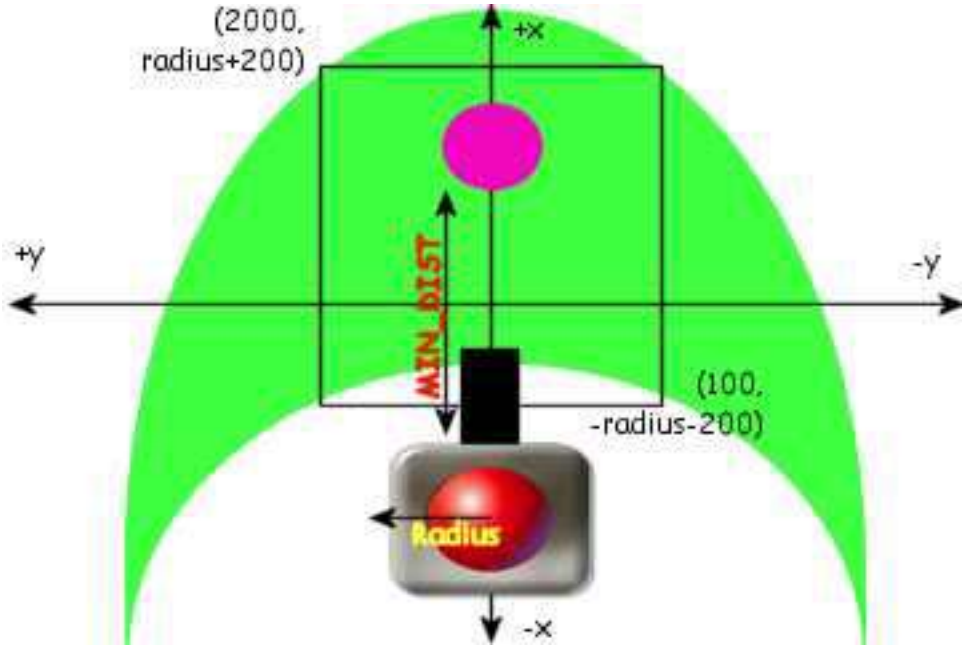
xRel = (blob.getXCG() - WIDTH/2.0) / WIDTH
yRel = (blob.getYCG() - HEIGHT/2.0) / HEIGHT

This formula calculates the relative position of the object on a XY plane with respect the origin (0, 0).



The resulting relative position provides us with the information to adjust the heading of the robot. If xRel is less than -.1 we need to turn to the left, likewise if xRel is greater than .1 we need to turn to the right.

The second sensory device used is the laser. The laser is used to determine how close the robot is while approaching a chair. We use Saphira and ARIA to extract the laser readings and determine when to halt the robot in front of the chair. The getClosetBox() function allows us to only concern ourselves with laser readings in a specific area around the robot. In this instance, we are only concerned with the area directly in front of the robot. Consider the following diagram:



The green field is the laser range. In this case, we have drawn an imaginary box to determine whether or not we are in front of a chair. When the distance from a chair is less than or equal to MIN_DIST, we halt the robot.

In release 2 we added localization and used this to get the global coordinates of detected chairs. To do this we use the ArTransform class. This class is initialized with the current position of the robot. After being initialized, this class can transform local robot coordinates to global map coordinates.

Chair Gripping Module

Assumptions:

- A chair is in the camera's view.
- Both colors are on the chair.
- The right side of the chair is cleared (this will be fixed, time permitting).
- The robot is within 900mm of the chair after Object Recognition module executes.

Algorithm Description:

The rafsGripChair module starts once the Object Recognition module has finished execution. Two conditions can be made at this point: the chair leg is directly in front, or it is not. The very first thing that is done in this module is to center the robot with the middle of the chair (the pink area). Once this is done, if the leg is aligned along the y axis within a given range (10 mm) of the middle of the chair, it will approach the chair until it is within a specified gripping distance. An example of an aligned chair is as follows:

At this point, it closes the gripper and sets the camera back to the original orientation, and completes. The complicated part comes in when the leg is not aligned correctly.

If the leg is not aligned within a given range (10 mm) of the middle of the chair (or pink), the robot will turn 90 degrees, and pans the camera angle to face back toward the chair. It will then drive in a turning fashion to maneuver around the chair, on the right side. Once the camera notices that the two colors are lined up along the y axis, it stops moving around the chair, and turns to face the chair. The camera then gets reset to be facing forward. Once this condition is met, the robot can continue to approach and grab the chair. An example of a chair not aligned correctly is as follows:

Other notes about the module:

It is important to note that the camera gets the center X, Y values for the color based on the camera's 2-dimensional view, in other words, whatever is in the camera's view has an X, Y coordinate value. The camera's X, Y coordinates are in no way related to the robots X, Y coordinates.

Object Avoidance Module

Assumptions:

- A chair has been found and gripped (Successful completion of Chair Gripping Module).
- We know the final destination's x, y coordinates (where we want release the chair).
- The X and Y coordinates of the robot's current point are different than the X and Y point the robot is currently located.

Algorithm Description:

The rafsObjAvoid module starts once the chair has been gripped. This module implements obstacle avoidance using the robot's laser. The laser fields are set up using the getClosestBox function which defines the corners of a box (field) to check for objects. A field is set up on each side of the robot, and if any object enters that field (or conversely if the robot travels in a direction where it is about to collide with an object), the robot adjusts its heading to avoid the

object and once clear of obstacles resets its heading to the nearest path to place the chair. The module finishes when the chair is placed in its proper (predefined) location (empty desk).

The robot will use inverse trigonometric (arc tangent) functions to determine a heading from where it is currently located to its goal position. Consider the following examples:

Assume the robot is at point (1, 1) and the goal point to place a chair is (26, 26). First find the difference between the start and end points. Delta X = 26-1 = 25, Delta Y = 26-1 = 25. Now the arc tangent function is at a (Delta Y/Delta X). The value returned is in radians, to get the value back into degrees it must be multiplied by 180/3.1416. The heading is then .7854 * (180/3.1416) = 45 degrees.

This works well for first quadrant headings. However when the Delta X value is less than zero (second and third quadrant values) 180 must be added to the answer to get a correct value relative to zero. As for the fourth quadrant (when Delta X is positive but Delta Y is less than zero) 360 could be added to get a corrected answer, however the robots software is "good enough" to accept a value like -45 degrees or 315 degrees.

A few notes of caution:

First: If the Delta X value is zero, the program would crash. If Delta Y is zero, the robot's heading will be either 90 degrees or 270 degrees. If Delta X is zero and Delta Y is positive, the heading is 90 degrees. If Delta X is zero and Delta Y is negative, the heading is 270.

Second: The algorithm provides for the case of Delta Y being zero. If Delta X is positive and Delta Y is zero, the heading will be 0 degrees. If Delta X is negative and Delta Y is zero, the heading is 180 degrees.

Lastly: This algorithm uses variables of type float for all calculations. It has been found that if type int is used Delta values can become overly truncated and "odd" values of 180, 0, 90, and 270 appear for no particular reason.

Other notes about the module:

Two functions are implemented in this module, but are not used. These two functions are farRight and farLeft which are both Boolean functions like nearRight and nearLeft, but testing revealed them to be redundant and were not used in the final algorithm. Later revisions to this module may use these functions.