

# 3D Localization Project

## Official Code Documentation

John Allard, Alex Rich  
2014 Summer Computer Science REU, Harvey Mudd College

July 10th, 2014

## Abstract

This is the official documentation for a series of interconnected programs written at Harvey Mudd during the summer of 2014 by John Allard and Alex Rich. The purpose of these programs is to successfully localize an actor in an environment using a preloaded 3D map, precomputed computer-vision related feature data, and a live image feed from an actor in the environment. The program starts with a large amount of guesses as to where the actor could be, and uses particle filtering to converge the particles upon the correct location in the environment. Read the research paper associated with this project for more information on the mathematics and other technicalities behind Monte Carlo Localization.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Pre-Localization . . . . .	3
1.2	During Localization . . . . .	3
<b>2</b>	<b>Perspective Generator</b>	<b>4</b>
2.1	Overview . . . . .	4
2.2	Errors and Successes . . . . .	4
2.3	Directory Tree . . . . .	4
<b>3</b>	<b>Database Generator</b>	<b>6</b>
3.1	Overview . . . . .	6
3.2	Directory Tree . . . . .	6
3.3	Issues . . . . .	6
<b>4</b>	<b>3D Localization</b>	<b>7</b>
4.1	Localization Program Overview . . . . .	7
4.2	File Structure . . . . .	7

# 1 Introduction

Accomplishing the task of successfully locating an actor in an environment using CV-derived features and a 3D model requires many individual steps. The success of the localization attempt depends strongly on the computation and data storage that is done before any attempt is made. With this in mind, we decided to subdivide the task of localizing an actor into three separate programs that must be run individually. Two of them are done before the localization attempt and are only needed to be ran once for each map that an actor is attempting to localize itself in. The 3rd program is the actual localization code that matches features to figure out where the actor is in the given map.

## 1.1 Pre-Localization

### 1. PerspectiveGenerator

- Takes a 3d model and a definition of a bounding box within the 3d map.
- Generates a grid within that bounding box, and makes a list of every grid intersection.
- Renders images from every point listed and saves them in a directory for feature processing.
- Has been tested with **Wavefront** .obj model files, should work with other types of files.

### 2. DatabaseGenerator

- Generates a database of computer-vision related feature data.
- This feature data can range from greyscale image, black and white image, SURF/SIFT/ORB feature data and descriptors, or even images that are contrast tuned.
- This program does the majority of the work and is the reason this Localization program can run efficiently in real time, it computes mounds of data and writes it all to file for future use.

## 1.2 During Localization

### 1. 3DLocalization Program

- The purpose of this program is to use the data generated by the **PerspectiveGenerator** and **DatabaseGenerator** program to localize an actor in real-time.
- Start by connecting to an actor and loading the pre-computed feature data about the environment.
- Generate a uniform distribution of particles over the map area, each particle represent a guess as to where the actor is.
- Take in a sensor reading from the robot in the form of an image. Process the image to extract feature data.
- Weigh the particles, generate a distribution according to the particle grouping and weighting.
- Sample from the distribution for new particles.
- Move the robot to a new location and move the particles a corresponding distance in the map.
- Iterate the loop from bullet 4 to 7.

## 2 Perspective Generator

### 2.1 Overview

1. General Purpose - The general purpose of this program is to convert a 3D model into a processable library of images that are rendered from different points in the model. This serves the purpose of allowing us to use existing 2D-Image feature detection algorithms (like Sped-Up Robust Feature Detection, or Scale Invariant Feature Detection) to match a 2D image feed from an actor to a point in a 3D model.
2. Dependencies - OpenGL 3.3, GLEW, GLUT, BOOST 1.46, OpenCV.

### 2.2 Errors and Successes

1. Very Important : The .vert and .frag Shader files must have the same name before the extension. This took me like 2-3 days to figure out and I did not see it anywhere inside the documentation for OpenGL.
2. The program will only work on more modern OpenGL version. Go to `consol` and type `glxinfo | grep "OpenGL version"` to find out which version is currently running on your computer. If it is not 3.3 exactly, this program might not run correctly. It would be useful to fix this but it's too lengthy of a process to concentrate on currently.
3. The 3D models are not normalized or scaled to proper real world scale. So 0.1 distance in the model might be 3 or 4 foot change in real life. So the program needs to have the `scale` variable adjusting according to each new model that the user tries to use.

### 2.3 Directory Tree

- /Localization/src/
  - Helper/
    - \* `HelperStructures.cpp` - Contains two helper structs for rendering data.
    - \* `MathHelp.h` - Contains math functionality and global state variables.
    - \* `MathHelp.cpp` - Definition of functionality defined in `MathHelp.h`.
  - IO/
    - \* `ProgramIO.h` - Declares all needed input and output for the program.<sup>1</sup>
    - \* `ProgramIO.cpp` - Definition of `ProgramIO.h`.
  - Main/
    - \* `main.cpp` - Starting points of the entire program. Initializes `OpenGL`, `GLUT`, and other dependencies.
  - Rendering/
    - \* `Render.h` - Declares all rendering functionality. Such as taking in the Vertex Array Object and texture data, calling any necessary translation or rotation commands, then rendering the map to the screen.
    - \* `Render.cpp` - Defines the functionality declared in `Render.h`.
  - Shaders/
    - \* `Shader.frag` - Fragmentation Shader file. Gets inserted into the `OpenGL` rendering and shading pipeline.
    - \* `Shader.vert` - Vertex Shader file. Gets inserted into the `OpenGL` rendering and shading pipeline.
    - \* `ShaderFunctions.h` - Declares all functions related to the shaders. Like reading the shader files and loading the shaders into `OpenGL`.

---

<sup>1</sup>Such as error logging, debugging code, or progress statuses

- \* ShaderFunctions.cpp - Defines the functionality declared in textttShaderFunctions.h.
  - View/
    - \* View.h
    - \* View.cpp
- /Localization/Inputfiles/
  - PointGenInput.txt - A sample input file for the program.
- /Localization/ProgramDesign/
  - InputFileFormat.txt - The format of the input file required to run this program.
- Localization/build/
  - All build files ignored, the program needs to be built on the computer it is going to be run upon.

## 3 Database Generator

### 3.1 Overview

1. Purpose - Creates a directory full of feature data to accompany each photo created by the Perspective Generator.
2. Features include:
  - Average Pixel Sum Image: Take original image, split into divisions, then for each grid square, take the average pixel intensity. This grayscale image is small, around 50x50.
  - Above Below Image: Take the average intensity of the Average Pixel Sum Image. If a pixel in that image is lighter, make it white. If it's darker, make it black. This image is the same size as the Pixel Sum Image but is binary.
  - Feature Descriptors: SURF and SIFT descriptors for extracted keypoints in each image.

### 3.2 Directory Tree

- DatabaseGenerator/src/
  - AverageImage.h - Contains function needed for computed features about images
  - CreateDB.cpp -
  - Similarity.h -
- DatabaseGenerator/build/
  - No build files will be pushed to Github

### 3.3 Issues

1. Needs some way to better filter keypoints.
2. Needs to accept more diverse features.

## 4 3D Localization

### 4.1 Localization Program Overview

1. Description

### 4.2 File Structure