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

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 a few 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 other group of programs are used for the actual localization attempt. This includes the actual 3DLocalization program and some optional GUI programs used for debugging and data-visualization purposes.

1.1 License and Info

- License - No license has been declared on this project, but feel free to download, modify, or distribute this code as you please. No warantee expressed or implied exists when you use or alter our code.
- Info - This project was written by John Allard and Alex Rich at Harvey Mudd College for the 2014 Computer Science REU program. Our mentor was the awesome Professor Zachary Dodds, and without whom this project would not have been possible to complete.
- Repository - The official repository for this program is `https://github.com/jhallard/3DLocalizer`.

1.2 Pre-Localization Programs Overview

This section will give a brief overview of the programs that need to be run prior to an attempt at localizing an actor in an environment.

1. PerspectiveGenerator
    - Takes an input file that specifies a user-supplied 3D model, the bounding box on the model, and some other information as input.
    - Generates a grid within that bounding box at a user defined spacing, and makes a list of every grid intersection.
    - Renders `[600x600]` images from every point enumerated in the list and saves them into the `/Data/RenderedImages/` directory.
    - Has been tested with `Wavefront .obj` model files, should work with other types of files.
    - The structure for the input file necessary to run this program is discussed in section 2 of this paper.
    - On a decent (by no means fast) computer it renders images at about 20 per second, so if you need to render 5000 images (as our group had to), it's going to take a little while. We did not have the available time to optimize this program.
2. DatabaseGenerator
    - Takes the name of a model directory inside of `/Data/RenderedImages/` that contains the rendered images from the model that the user would like to generate feature data for as a command line argument.
    - Generates a database of computer-vision related feature data from the images rendered by the `PerspectiveGenerator` program.
    - 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 using many different types of computer vision algorithms. It then organizes and writes all of this data to the `/Data/FeatureData/` directory for use by the Localization program.

1.3 Localization Programs Overview

This section will give a brief overview of the program directly involved in the localization attempt, as well as some optional-to-run programs to expand our testing and data-visualization capabilities.

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.
   - This program starts by loading the data from the `3DLocalization/Data/FeatureData/` directory into live variables. The most computationally expensive of these processes is generating a master map that returns feature data given a certain perspective in the map.
   - Establish a connection with the robot/actor :
     (a) Then it begins publishing data under our `ros::Publisher` object and waits for the robot control program to subscribe to this publisher.
     (b) When a subscriber is detected, a handshake code is sent out over our data publisher, letting the robot know that we are ready to begin the localization process.
     (c) Next, the localization program searches for the robots data publisher and subscribes to it when found. This connection will allow our algorithm to know the movement of the robot and other information to help in the localization process.
   - Begin the main localization loop :
     (a) Generate a uniform distribution of particles over the map area, each particle represent a guess as to where the actor is.
     (b) Take in a sensor reading from the robot in the form of an image. Process the image to extract feature data.
     (c) Weigh the particles, generate a distribution according to the particle grouping and weighting.
     (d) Sample from the distribution for new particles.
     (e) 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. TestRobot Program

   - This program is designed to act as a 'virtual' robot to aid in the testing and demonstration of our localization program. It acts the same as a regular robot control program except it traverses the virtual 3D model of the environment as send rendered images to our localization program instead of acting in the real 3D environment.

3. GUI Programs

   (a) PyViewer Program
   - Visualizes the particles in the current map from a 'birds eye view'. Displays their position and orientation in the map and colors them in a heat-map style according to their probability of being the correct location for the robot.
   - Mainly used for debugging purposes and to generate images for slides and posters.

   (b) MapViewer Program
   -

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 Program Flow

1. Before you can run the program, you must first build it by going into the `/PerspectiveGenerator/build/` directory and entering `cmake ..` then `make` . This will generate a runnable file called `./PerspectiveGenerator` inside of the same directory.

2. Create a directory containing your 3D model, `.mtl` files, and texture files. Create a new directory with a meaningful name in the `/3DLocalizer/Data/ModelData/` directory. Name this directory something meaningful because you will need to pass in this directory name as input to a few programs in this project.

3. You will then need to hand type an input file for the program, and place it in the `PerspectiveGenerator/Inpufiles/` directory. This file must contain the following, line by line.

   - Name of the directory containing your 3D model file inside of the `/3DLocalizer/Data/ModelData/` directory.
   - Name of the 3D model file inside the above folder (e.g. `dog.obj`).
   - Bounding box for your 3D model in the following format :
     `xmin xmax ymin ymax z gridDensity dtheta`.

   As you can see, the height z must be kept contant for this program. This will hopefully be changed in the future. The gridDensity value determines the spacing between consecutive image renderings, it should be in units according to the scale of your 3D model, not the real life space that the actor is in. The dTheta detemines the angle we rotate by when generated images from a single point with different perspectives (e.g. a value of 30 degrees will cause the program to render 12 images at `0, 30, 60, ..` degrees from every `[x y z]` point on the grid.

4. From the `/PerspectiveGenerator/build/` folder, call the `./PerspectiveGenerator` executable and pass in the name of the input file as an argument.

5. The program will take 10 seconds or so to load, when it does you will see a window with a rendering from a single point in the map.

6. Click anywhere on the screen and the program will automatically go through all points generated inside your map and render images to the `/Data/RenderedImages/` folder under the same name of your folder containing the model data.

7. When the program stops scrolling through the images (at about 20-25 per second), you can close the program.

2.3 Issues and Fixes

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 console 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.
4. Sometimes this program runs unbelievably slow for a short while, while it normally runs at a pretty decent rate. I think it has something to do with the older workstation I'm using, it almost seems as if some of the time it is attempting to use the CPU to do the graphics computations.
5. It is inconvenient to force the program to render to the window and then take the image from the window to render to the file, and this is obviously not the best programming practice. However, this program was merely seen as a stepping stone to the goal of localizaing a robot and I happen to be quite new to OpenGL, so we had to take version of this program that actually worked.
6. *Feel free to help make this program better!* - Contact me on Github (@jhallard) for contribution rights to master branch.

2.4 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.[1]
    - `ProgramIO.cpp` - Defintition 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 the the screen.
    - `Render.cpp` - Defines the functionality declared in texttt 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`.
    - `ShaderFunctions.cpp` - Defines the functionality declared in textttShaderFunctions.h.
  - `View/`
    - `View.h` - This file declares all functions related to the view matrix for the program. The view matrix determines the properties of the camera, like the field of view, near and far points, etc.

[1]Such as error logging, debugging code, or progress statuses

    * `View.cpp` - Contains the definitions of all global variables and functions declared inside of the `View.h` file.

- `/Localization/Inputfiles/`
  - `PointGenInput.txt` - A sample input file for the program. This file only works with the Sprague2ndFloor `.obj` file and is only left here as a concrete example of the InputFile formatting for this 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. Dependencies - OpenCV, Boost 1.46.

3.2 Program Flow

1. Navigate to the `PreLocalization/DatabaseGen/build/` folder via the terminal.
2. Type `cmake ..` followed by `make` to build the program. If error messages appear, check in *Section 3.3, Issues and Fixes* for more information. If nothing is there, feel free to open up a bug reguest on Github.
3. From the same `/build/` folder, type `./CreateDB ModelName` , where `yourModelNameHere` is replaced by the name of the directory that contains your 3D model data inside of the `3DLocalization/Data/Mode` directory.
4. Press enter and the program will run. It will start by loading the images from the `3DLocalization/Data/Rende` directory from file into active memory. A terminal message will be displayed when this is finished (can take up to a few minutes to load 10,000 images).
5. The program will loop through each image and calculate a set of feature data for each image, the set of feature data contains the following :
   - Features derived from the Speeded-Up Robust Features (SURF) algorithm.
   - Feaures derived from the Scale Invariant Feature Transform (SIFT) algorithm.
   - 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.
6. These features will be written to file inside of the `/3DLocalization/Data/FeatureData/` directory under a folder with the same as the directory containing the 3D model data. The individual files that are written will consist of `.jpg` and `.yml` file, and the name will be the `[x y z dx dy dz` coordinates of the perspective that the feature data corresponds to.
7. A progress bar will be displayed via the terminal letting you know how long the process will take. It has taken us 17 minutes to compute and store the feature data corresponding to 6500 rendered `[600x600]` images.

3.3 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

3.4 Issues and Fixes

1. Needs some way to better filter keypoints.

2. Needs to accept more diverse features.

4 3D Localization

4.1 Localization Program Overview

As described earlier, this program is the one that actually uses the Monte Carlo Localization (MCL) algorithm and computer-vision derived feature data to localize an actor in an environment. As also was described earlier, this program can run after all of the pre-localization requirements have been met[2]. An abstract description of this program would be as follows :

```
1.)  Load data necessary to localize the actor and generate guesses about the actors position.
2.)  Connect to the actor via some combination of callback functions.
3.)  Loop through the MCL algorithm until your guesses converge on a location.
```

Asside from these main points, we have also designed a few graphical programs to help illustrate the process to the user. The help with both debugging and gaining a better understanding of what is happening inside of the program.

The coding goal of this program was to use practices that are generally accepted to be good[3]. This means abstracting the problem of localizing a robot into a set of classes and a set of independent functions. We also had to make decisions about adding a GUI to the program, or which libraries we wanted to use for callback functionality. Our solutions to the problems will be documented and illustrated in the following sections. All code is lightly documented inline as well as documented inside of this paper. All changes made are consistantly push to Github at our official repository[4], and thus an entire history of our code is available online.

4.2 Program Flow

This section will only provide an overview of how the program runs and will avoid the technical details. A more detailed description of the parts of this program are discussed later in this document.

1. Build the program by following the instructions in *Section 4.3, Build the Program.*
2. Starting the Program
   - Navigate to the `/Localization/build/` directory via the terminal.
   - Type `./3DLocalization modelName` and replace `modelName` with the name of the directory that contains your 3D model, press enter to start the program.
   - The program will attempt to load the feature data associated with the model, this should take under a minute for about 7000 images. When it is finished, the program will pause to let you start your robot control sequence.
3. Connecting to the Actor/Robot
   - Have your actor control program search for `ros::Publisher` under the name `MCL_DATA_PUBLISHER`. This publisher will publish data of type `ros::std_msgs::String` . The recieved string must be split into tokens based on a `_` character.
   - When found, subscribe to said publisher with a `ros::Subscriber` object.
   - Subscribing to our publisher will trigger our program to send a handshake command to your robot. The localization program will then search for the two robot data publishers or go into a loop to wait for them to become live.
   -
   - More information on this process is covered in *Section 4.5, Robot Interface.*
   -

[2]For more information on these, see sections 2 and 3.

[3]Practices like Object Oriented Programming, avoiding go-to statments, proper naming conventions, and inline code documentation

[4]https://Github.com/jhallard/3DLocalization

4.3 Building the Program

- Dependencies - Boost 1.46, OpenCV, ROS Hydro, Catkin Libraries for ROS.
- Performing the build
  - Ensure that you have all dependencies listed above.
  - Navigate to the `/Localization/build/` folder via the terminal.
  - Enter `cmake ..` follow by `make` into the terminal to build the program.
- If any errors appear in the previous steps please see the *Errors and Issues* section.

4.4 Classes

-

4.5 Robot Interface

-

4.6 GUI Programs

1. MapViewer - View the active particles of the program inside of the 3D model as they are generated and filtered from the program. The best guess for the robots position is also shown.
2. PyViewer - View the active particles of the program from a birds eye view above the map. Particles are colored in a heat-map style according to likelyhood of being the location of the robot.

4.7 Errors and Issues

Official error log can be found at this projects repository on Github.

1.

4.8 Directory Tree

- `/Localization/src/` -
  - `Boot/`
    - `Boot.h` - Loads feature data into memory and performs a 'hand-shake' with the actor in the evironment.
  - `GUI/`
    - TODO
    - TODO
  - `HelperStructs/`
    - `Characterizer.h` - Defines a data structure to hold different types of features associated with a certain perspective.
    - `Perspective.h` - A data structure that defines a perspective in the enrivonment. In short, a `[x y z]` vector for position and a `[dx dy dz]` vector for camera orientation.
    - `Globals/`
      - `Globals.h` - Contains the declaration of the two globals needed for the program : The list of perspectives and the std::map that maps each perspective to a specific collection of feature data about that perspective.
      - `Globals.cpp` - Definition of the globals declared in the above file.
  - `IO/`

    - `ProgramIO.h` - Declares a list of functions to allow interaction with the user via terminal messages.
    - `ProgramIO.cpp` - Defines the functions declared in `ProgramO.h`.
  - `MCL/`
    - `ActiveParticles/`
      - `ActiveParticles.h` - A data structure that wraps the vector of particles needed to run the MCL algorithm with useful functionality and data validation tests. Handles creating a distribution and sampling from that distribution to refill the vector of particles.
      - `ActiveParticles.cpp` - Definition of the functionality defined in `ActiveParticles.h`.
    - `Control/`
      - `Controller.h` - The master control object for our program. Contains the vector of particles and the data structure that represents the current state of the robot attempting to be localized. The `main()` function only needs to call `Controller::SpinOnce()` to execute an entire iteration of the MCL algorithm.
      - `Controller.cpp` - Definition of the functionality defined in `Controller.h`.
    - `Matching/`
      - `Matching.h` - A file that holds functions that compute relations between different images using computer vision algorithms (e.g. SURF, SIFT).
  - `Main/`
    - `Main.cpp` - The official starting point of the program. Calls the boot functions, initializes the Controller class, and starts the main localization loop.
  - `Particle/`
    - `Particle.h` - Declaration of the `Particle` class. This data structure represents a single guess as to where the robot could currently be in the environment.
    - `Particle.cpp` - Definition of all of the functionality declared in the `Particle` class.
  - `Robot/`
    - `RobotState.h` - Declaration for the `RobotState` class. This class holds a model of a general robot and functions to do work on the model.
    - `RobotState.cpp` - Implementation of the `RobotState` class.

- `/Localization/TestRobot/` -
  - `main.cpp` - The entire control program for the test robot. Contains the callback functions and the main function.
  - `package.xml` - Package to build ROS dependencies for the test robot.
- `/Localization/build/` - Contains build files for the program. Will contain a .gitignore to ignore any build files from being pushed to the repository.
- `/Localization/README.md` - Quick breakdown of the 3DLocalization program.
- `/Localization/CMakeLists.txt` - Tells CMake how to create the program and link dependencies.