3D Point Cloud Map Generation with Choreonoid

AY2021

Isuru Jayarathne

Setting up Choreonoid environment

- System requirements
  - Ubuntu 20.04
- Clone the latest version of Choreonoid from GIT repository
  - git clone https://github.com/choreonoid/choreonoid.git
- Execute following commands to build the code and install
  - cd choreonoid
  - misc/script/install-requisites-ubuntu-20.04.sh
  - mkdir build
  - cd build
  - cmake ..
  - make
  - sudo make install

Creating a Choreonoid project

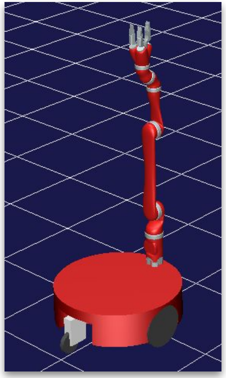
- Create a folder (aizu_wheel_mapping) in the “ext” folder in Choreonoid source folder.
- Create two folders named “models” and “src” in the created folder.
- Copy “CMakeLists.txt” file from “ext” folder to the project folder (aizu_wheel_mapping)
- Create two folders named “kinect” and “vlp” to save point cloud data from kinect camera and LiDAR sensor.
- Folder structure can be seen in the figure.
- Complete project can be found here
  https://github.com/ijmax/Choreonoid_mapping

```
Choreonoid
|-- ext
|    |-- aizu_wheel_mapping
|    |    | -- src
|    |    |     |-- CMakeLists.txt
|    |    |     |-- *.cpp
|    |    |-- models
|    |    |    |-- AizuWheel
|    |    |    |-- Labo1
|    |    |-- kinect
|    |    |-- vlp
|    |    |-- CMakeLists.txt
|    |-- CMakeLists.txt
```

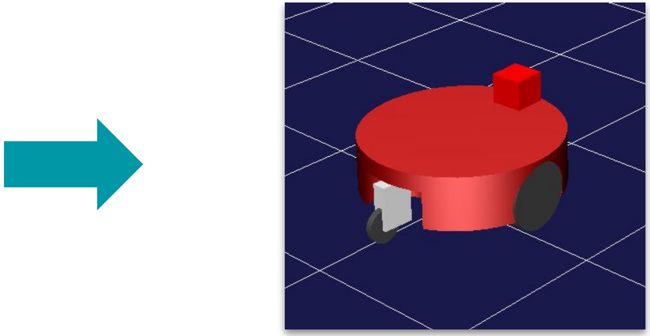
Folder structure

Customizing existing robot model

- Copy the folder “AizuWheel” from following location into the “models” folder
  - /usr/local/share/choreonoid-1.8/model/
- Edit “AizuWheel.body” file by replacing “JACO2.body” with “box.body” in line 136
- Change the translation values into [0.25, 0, 0.16] in line 139



Default model



Modified model

Creating a object with depth sensors

- Goto the “AizuWheel” folder and create new file named “box.body”
- Adding a Kinect camera
  - Type is “Camera” and format is “COLOR_DEPTH”
  - Outputs x,y,z coordinates and R,G,B values of a point
  - Resolution can be changed by fieldOfView value, width, and height
- Adding LiDAR sensor
  - Type is “RangeSensor”
  - Horizontal point resolution can be changed by “yawRange”, and “yawStep”
  - Vertical point resolution can be changed by “pitchRange” and “pitchStep”
- Other necessary parameters can be seen in the code snippet
  - Complete code can be found here https://github.com/ijmax/Choreonoid_mapping/blob/main/models/AizuWheel/box.body

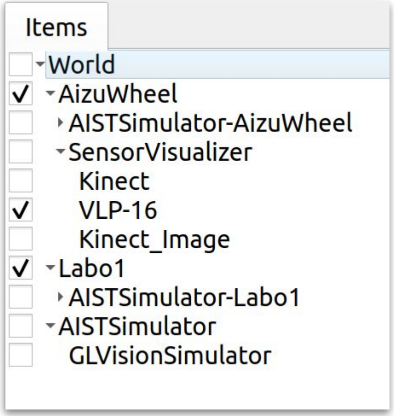
```
type: Camera
      name: Kinect
      translation: [ 0.0, -0.10, 0.10 ]
      rotation: [ [ 1, 0, 0, 90 ], [ 0, 1, 0, 180 ] ]
      format: COLOR_DEPTH
      fieldOfView: 62
      width: 320
      height: 240
      frameRate: 30
      on: true

type: RangeSensor
      name: VLP-16
      translation: [ 0, 0, 0.1 ]
      rotation: [ [ 1, 0, 0, 90 ], [ 0, 1, 0, -90 ] ]
      yawRange: 360.0
      yawStep:  0.4
      pitchRange: 30.0
      pitchStep: 2.0
      scanRate:  5
      maxDistance: 100.0
      on: true
```

Creating a Choreonoid project

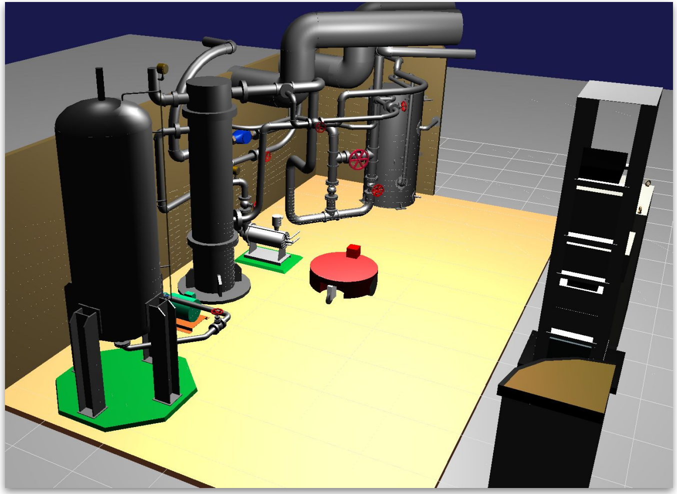
- Open Choreonoid and save the project (File -> Save Project) in created project folder (aizu_wheel_mapping).
- Create a “World” by selecting File -> New -> World
- Load “AizuWheel.body” in the project folder by selecting File -> Load -> Body
- Also, load “Labo1.body” from the Choreonoid installation folder (/usr/local/share/choreonoid-1.8/model/Labo1)
- Create an “AISTSimulator” under the “World” object by selecting File -> New -> AISTSimulator
- Create a “GLVisionSimulator” under the AISTSimulator by selecting File -> New -> GLVisionSimulator
- Change the value of “Record vision data” of “GLVisionSimulator” to true in the properties

Creating a Choreonoid project (Cont.)

- Create a “SensorVisualizer” under the “AizuWheel” item by selecting File -> New -> SensorVisualizer
- Check the sensor(s) under the “SensorVisualizer” that need to be appeared in the scene view.
- After starting the simulator, point cloud created by sensor can be seen as white dots.



Items tree



Scene view

Creating a controller for move the robot

- Create a file named "wheel_controller.cpp" in the "src" folder.
- Add following code which follows the code format in the Tank tutorial (https://choreonoid.org/en/manuals/latest/simulation/tank-tutorial/step4.html)
- The robot can be controlled using the virtual joystick (I,J,K,L buttons).
- Complete code can be found here https://github.com/ijmax/Choreonoid_mapping/blob/main/src/wheel_controller.cpp

```
class AW_Controller : public SimpleController
{
  Link* trackL;
  Link* trackR;
  Joystick joystick;

public:
  virtual bool initialize(SimpleControllerIO* io) override
  {
    trackL = io->body()->link("L_WHEEL");
    trackR = io->body()->link("R_WHEEL");

    io->enableOutput(trackL, JointVelocity);
    io->enableOutput(trackR, JointVelocity);

    return true;
  }

  virtual bool control() override
  {
    static const int axisID[] = { 2, 3 };

    joystick.readCurrentState();

    double pos[2];
    for(int i=0; i < 2; ++i){
      pos[i] = joystick.getPosition(axisID[i]);
      if(fabs(pos[i]) < 0.2){
        pos[i] = 0.0;
      }
    }

    double k = 2.0;
    trackL->dq_target() = k * (-2.0 * pos[1] + pos[0]);
    trackR->dq_target() = k * (-2.0 * pos[1] - pos[0]);

    return true;
  }
};
```

Creating a controller to save point clouds

- Create a file named “kinect_data_recoder.cpp” in the “src” folder.
- PCD files were saved after transforming them using following code.
- A PCD file was saved in the given folder after each button (B) click of the virtual joystick.
- PCL library is required to be installed to compile this controller.
- Make sure PCL library has been installed before compilation.
- Complete code can be found here
  https://github.com/ijmax/Choreonoid_mapping/blob/main/src/kinect_data_recorder.cpp

```
void savePCD()
  {
    const Image& imgData = knt->constImage();
    const unsigned char* pixels = imgData.pixels();

    const int width  = imgData.width();
    const int height = imgData.height();

    pcl::PointCloud<pcl::PointXYZRGB> cloud;
    cloud.width    = width;
    cloud.height   = height;
    cloud.is_dense = false;
    cloud.points.resize(cloud.width * cloud.height);

    size_t i = 0;
    size_t ci = 0;
    for(const auto& e: knt->constPoints()) {
      if (e[1]<2 and e[1]>-0.5)
      {
        cloud[i].x = e(0);
        cloud[i].y = e(1);
        cloud[i].z = e(2);
        cloud[i].r = pixels[3*ci + 0];
        cloud[i].g = pixels[3*ci + 1];
        cloud[i].b = pixels[3*ci + 2];
        ++i;
      }
      ++ci;
    }

    Eigen::Affine3f transform = Eigen::Affine3f::Identity();

    Position pos = ioBody->rootLink()->position();
    const Vector3 t = -pos.translation();
    Vector3 r = rpyFromRot(pos.rotation());
    transform.translation() << t.y(), t.z(), t.x();
    transform.rotate (Eigen::AngleAxisf (r[2],
Eigen::Vector3f::UnitY()));

    pcl::PointCloud<pcl::PointXYZRGB> transformed_cloud;
    pcl::transformPointCloud (cloud, transformed_cloud, transform);

    pcl::io::savePCDFileBinaryCompressed ("data/cloud" +
to_string(counter) + ".pcd", transformed_cloud);
    (*os) << "Saved a pcd file" << std::endl;
  }
```

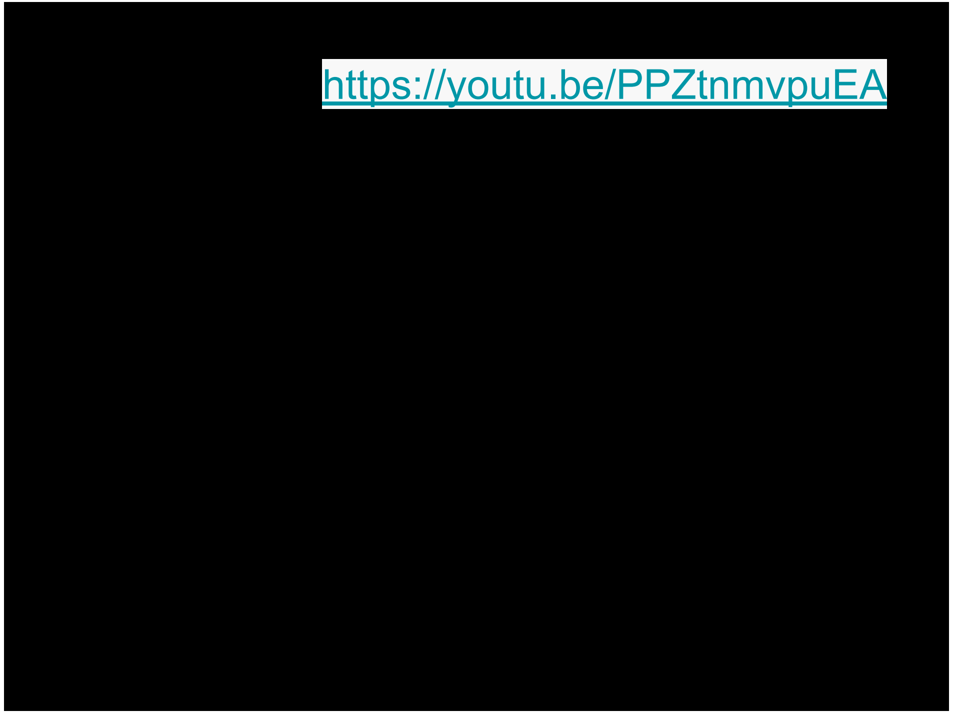
Compiling and adding controllers

- Create a “CMakeLists.txt” in the “src” folder and add the following code. https://github.com/ijmax/Choreonoid_mapping/blob/main/src/CMakeLists.txt
- Compile the Choreonoid source in order compile controller codes
    - Goto “build” folder in the Choreonoid source folder.
    - Execute “cmake ..” then “make” commands
- Create a “SimpleController” under the “AizuWheel” in the Choreonoid project and name it as “WheelController”.
- Set the compiled controller in the field “Controller module” in the properties list.
- Compiled controller module can be found in <Choreonoid source>/lib/choreonoid-1.8/simplecontroller/AW_wheel_controller.so.
- Add another “SimpleController” and name it as “KinectController”.
- Set the “Controller module” as same as previous step (<Choreonoid source>/lib/choreonoid-1.8/simplecontroller/kinect_data_recorder)

```
find_package(PCL REQUIRED common io surface features)

include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})

add_cnoid_simple_controller(AW_wheel_controller wheel_controller.cpp)
add_cnoid_simple_controller(kinect_data_recorder kinect_data_recoder.cpp)

target_link_libraries (kinect_data_recorder ${PCL_COMMON_LIBRARIES} ${PCL_SURFACE_LIBRARIES} ${PCL_FEATURES_LIBRARIES} pcl_io)
```

Simulation videos

AizuWheel in the Labo1



AizuWheel in the LiCTIA 1st floor

Merging recorded point clouds

- Python script has been use merge and downsample the recorded point clouds.
- Install necessary libraries using “pip” command.
- Pip install open3d
- Execute python file by passing point clouds folder path as a console parameter.
- Merged point could is generated in the same folder as python script is in.
- Downsampling factor can be adjusted to change the resolution of the final point cloud.

```
import open3d as o3d
import os
import sys
import numpy as np

path = sys.argv[1]
print("path: " + path)

if (path == ""):
	path = "."

files = os.listdir(path)

pcd0 = o3d.io.read_point_cloud(path + "/" + files[0])

point_array = np.asarray(pcd0.points)
color_array = np.asarray(pcd0.colors)

for i in range(len(files)):

	if (i>0):
		pcd = o3d.io.read_point_cloud(path + "/" + files[i])
		ps = np.asarray(pcd.points)
		cs = np.asarray(pcd.colors)
		point_array = np.concatenate((point_array, ps))
		color_array = np.concatenate((color_array, cs))

merged = o3d.geometry.PointCloud()
merged.points = o3d.utility.Vector3dVector(point_array)
merged.colors = o3d.utility.Vector3dVector(color_array)
pcd_down = merged.voxel_down_sample(voxel_size=0.05)
o3d.io.write_point_cloud("merged_cloud.pcd", pcd_down)
```

Merged point cloud