Individual Lab Report

Bikramjot Singh Hanzra
Team B – Auto Pirates
Teammates – Tushar Chugh, Shiyu Dong, Tae-Hyung Kim, William Seto
ILR03

October 30, 2015

Contents

List of Figures

1 Individual Progress

In the last week, I worked upon on the following tasks -

- Building the model of the boat in SolidWorks
- Building the SBPL library and running the example codes
- Setting up Gazebo and building a sample environment

In the subsections below, I have described in detail the tasks I have completed during the last week.

1.1 Building the model of the boat in SolidWorks

I clubbed my SolidWorks assignment with the project. The idea was to export the SW Assembly into the ROS compatible URDF format using `sw_urdf_exporter` [1] add-on for SolidWorks. The URDF file will then be used to render the boat in the simulator. It took me three days to finish this task. I designed 8 parts of the total 10 parts assembled myself. The remaining 2 parts – LIDAR and engine were downloaded from GrabCAD.

Figure 1 shows the isometric view of the boat.

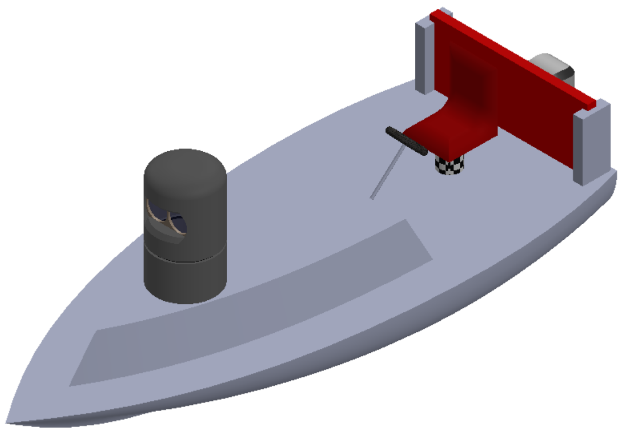

Figure 1: Isometric View of the boat

Figure 2 shows the exploded view of the boat.

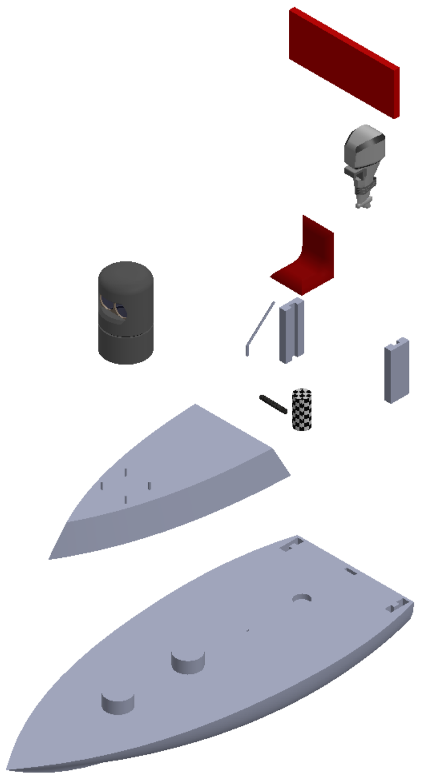

Figure 2: Exploded View of the boat

1.2 Building the SBPL library and running the example codes

The second task, I worked on during the past week was building the SBPL [2] library from source. After successfully building the library, I prepared the demo for the Progress Review 2. Figure 3 shows the path planned by the planner in the sample Willow Garage indoor map provided in the examples of SBPL. Given the initial and final positions, the path planner gives us the x, y and θ at each time unit to reach the destination as shown –

x ——- y ——- θ
0.113 0.113 0.000
0.110 0.113 0.000
0.107 0.113 0.000
0.104 0.113 0.000
0.101 0.113 0.000
0.099 0.113 0.000
0.096 0.113 0.000
0.093 0.113 0.000
0.090 0.113 0.000

```
0.088 0.113 0.000
0.085 0.113 0.000
0.082 0.113 0.000
0.079 0.113 0.000
0.076 0.113 0.000
0.074 0.113 0.000
0.071 0.113 0.000
0.068 0.113 0.000
0.065 0.113 0.000
0.062 0.113 0.000
0.060 0.113 0.000
```

Listings 1, 2 and 3 show the code that was used to prepare the demo. I wrote listings 1 and 2 myself. Listings 3 was downloaded from the SBPL website.

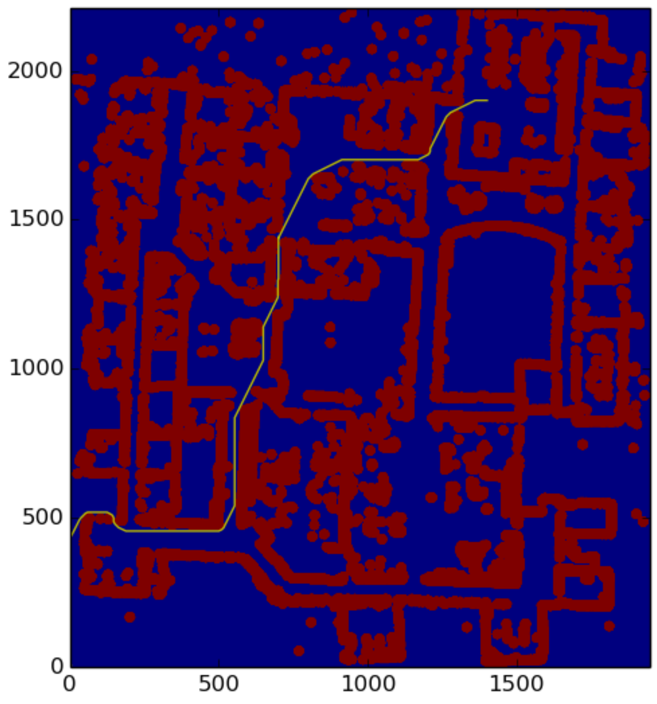

Figure 3: Path generated by the planner

1.3 Setting up Gazebo and building demo environment

I also skimmed through the Gazebo [3] tutorials and got a very good idea of how the simulator works. Figure 4 shows a sample environment that I created by dragging and dropping the available models that come with Gazebo to create a world. This world can be saved in a file and can be reloaded again whenever needed.

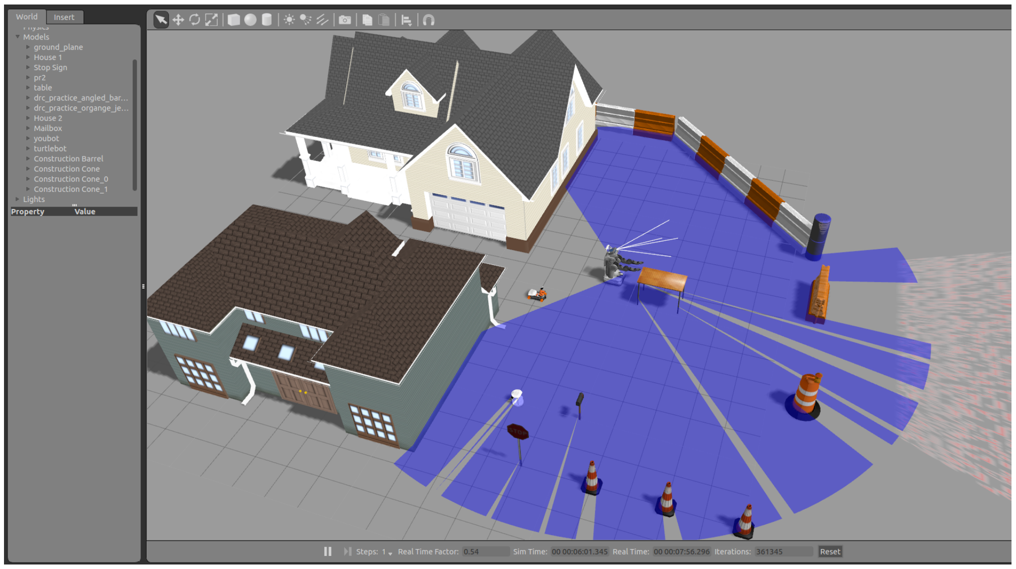
Figure 4: Sample environment created using Gazebo simulator

2 Challenges

One of the biggest downside of SBPL is that it is not well documented. So, it was pretty hard to understand the API. The explanation in the tutorials was terse and it was hard to grasp all the details without referring to other materials online. After building the example code, the example code cannot be run using `rosrun` command. The executable is created in the `build` directory of the workspace and `rosrun` is unable to find it. It took me a while to figure this out.

While designing the boat in SolidWorks, I spend a considerable amount of time in building the lofted base of the boat. This part was not difficult to implement but since I have only elementary proficiency in SolidWorks, I had to first watch some YouTube videos to get it done.

In the Gazebo simulator, although I understood the philosophy behind the simulator, I wasn't particularity comfortable with the xml code for the models and worlds. I need to put more effort into this section.

3 Teamwork

The work done by the rest of team is as under –

- Tushar Chugh – Tushar told me that he studies graph algorithm. He also built the SBPL library after me. He also worked on the *PCB Conceptual Design* and revising the sections for the Systems Engineering Presentation.
- Shiyu Dong – Shiyu read this paper – *Collision Avoidance for Vessels using Low-Cost Radar Sensor*. He then explained this to me and William. Shiyu also worked on updating the *Risk Management*, and *Fall Validation Experiment* sections for Systems Engineering Presentation.

- Tae-Hyung Kim – Kim used the SBPL library to generate a grid occupancy map of a sample environment in `stage` simulator. He also conversed with the staff of the SBPL lab to get information on the SBPL package.
- William Seto – William did an **awesome job** on interfacing OpenCPN with ROS. He also updated the *Description*, *Requirements*, and *Use Case Study* sections for Systems Engineering Presentation.

4 Work Overview for Coming Week

In the next week, I will be concentrating on –

- Logging real data from the RADAR using ROS. In the demo last week, we generated fake data using OpenCPN to show the interfacing with RADAR.
- Start working on code for preprocessing the radar data to remove noise.
- Start working on Assignment 12 after feedback from Luis and Ander.

5 Appendix 1

Sources Code for the examples are attached in this section.

```
rm -rf sol.txt
rm -rf sol.txt.discrete
./xytheta willow-25mm-inflated-env.cfg pr2.mprim
cat sol.txt
python visualize.py willow-25mm-inflated-env.cfg sol.txt.discrete
```

Listing 1: Source Code for the bash script for running the `xytheta` executable

```
# CMakeList file for compiling the package
cmake_minimum_required(VERSION 2.8.3)
project(pirates-planning)

find_package(catkin PkgConfig REQUIRED)
pkg_check_modules(SBPL REQUIRED sbpl)
include_directories(${SBPL_INCLUDE_DIRS})
link_directories(${SBPL_LIBRARY_DIRS})
add_executable(xytheta src/xytheta.cpp)
target_link_libraries(xytheta ${SBPL_LIBRARIES})
catkin_package(
)
```

Listing 2: Source Code of `CMakeLists.txt`

```
#include <cstring>
#include <iostream>
#include <string>
#include <sbpl/headers.h>

using namespace std;

// creating the footprint
void createFootprint(vector<sbpl_2Dpt_t>& perimeter){
    sbpl_2Dpt_t pt_m;
    double halfwidth = 0.01;
    double halflength = 0.01;
    pt_m.x = -halflength;
    pt_m.y = -halfwidth;
    perimeter.push_back(pt_m);
    pt_m.x = halflength;
    pt_m.y = -halfwidth;
    perimeter.push_back(pt_m);
    pt_m.x = halflength;
    pt_m.y = halfwidth;
    perimeter.push_back(pt_m);
    pt_m.x = -halflength;
    pt_m.y = halfwidth;
    perimeter.push_back(pt_m);
}

void initializeEnv(EnvironmentNAVXYTHETALAT& env,
                   vector<sbpl_2Dpt_t>& perimeter,
                   char* envCfgFilename, char* motPrimFilename){
    if (!env.InitializeEnv(envCfgFilename, perimeter, motPrimFilename)) {
        printf("ERROR: InitializeEnv failed\n");
        throw SBPL_Exception();
    }
}

void setEnvStartGoal(EnvironmentNAVXYTHETALAT& env,
                     double start_x, double start_y, double start_theta,
                     double goal_x, double goal_y, double goal_theta,
                     int& start_id, int& goal_id){

    start_id = env.SetStart(start_x, start_y, start_theta);
    goal_id = env.SetGoal(goal_x, goal_y, goal_theta);
}

void initializePlanner(SBPLPlanner*& planner,
                       EnvironmentNAVXYTHETALAT& env,
                       int start_id, int goal_id,
                       double initialEpsilon,
                       bool bsearchuntilfirstsolution){
    // work this out later, what is bforwardsearch?
    bool bsearch = false;
    planner = new ARAPlanner(&env, bsearch);

    // set planner properties
    if (planner->set_start(start_id) == 0) {
        printf("ERROR: failed to set start state\n");
        throw new SBPL_Exception();
    }
    if (planner->set_goal(goal_id) == 0) {
        printf("ERROR: failed to set goal state\n");
        throw new SBPL_Exception();
    }
    planner->set_initialsolution_eps(initialEpsilon);
    planner->set_search_mode(bsearchuntilfirstsolution);
}

int runPlanner(SBPLPlanner* planner, int allocated_time_secs,
               vector<int>&solution_stateIDs){
    int bRet = planner->replan(allocated_time_secs, &solution_stateIDs);

    if (bRet)
        printf("Solution is found\n");
    else
        printf("Solution does not exist\n");
    return bRet;
}

void writeSolution(EnvironmentNAVXYTHETALAT& env, vector<int> solution_stateIDs,
                   const char* filename){
    std::string discrete_filename(std::string(filename) + std::string(".discrete"));
    FILE* fSol_discrete = fopen(discrete_filename.c_str(), "w");
    FILE* fSol = fopen(filename, "w");
    if (fSol == NULL) {
        printf("ERROR: could not open solution file\n");
        throw SBPL_Exception();
    }
    // write the discrete solution to file
    for (size_t i = 0; i < solution_stateIDs.size(); i++) {
        int x, y, theta;
        env.GetCoordFromState(solution_stateIDs[i], x, y, theta);
        double cont_x, cont_y, cont_theta;
        cont_x = DISCXY2CONT(x, 0.1);
        cont_y = DISCXY2CONT(y, 0.1);
        cont_theta = DiscTheta2Cont(theta, 16);
        fprintf(fSol_discrete, "%d %d %d\n", x, y, theta);
    }
    fclose(fSol_discrete);
    // write the continuous solution to file
    vector<sbpl_xy_theta_pt_t> xythetaPath;
    env.ConvertStateIDPathintoXYThetaPath(&solution_stateIDs, &xythetaPath);
    for (unsigned int i = 0; i < xythetaPath.size(); i++) {
        fprintf(fSol, "%.3f %.3f %.3f\n", xythetaPath.at(i).x,
                                          xythetaPath.at(i).y,
                                          xythetaPath.at(i).theta);
    }
    fclose(fSol);
}

void planxythetalat(char* envCfgFilename, char* motPrimFilename){
    // set the perimeter of the robot
    vector<sbpl_2Dpt_t> perimeter;
    createFootprint(perimeter);

    // initialize an environment
    EnvironmentNAVXYTHETALAT env;
    initializeEnv(env, perimeter, envCfgFilename, motPrimFilename);

    // specify a start and goal state
    int start_id, goal_id;
    setEnvStartGoal(env, .11, .11, 0, 35, 47.5, 0, start_id, goal_id);

    // initialize a planner with start and goal state
    SBPLPlanner* planner = NULL;
    double initialEpsilon = 3.0;
    bool bsearchuntilfirstsolution = false;
    initializePlanner(planner, env, start_id, goal_id, initialEpsilon,
                      bsearchuntilfirstsolution);

    // plan
    vector<int> solution_stateIDs;
    double allocated_time_secs = 10.0; // in seconds
    runPlanner(planner, allocated_time_secs, solution_stateIDs);

    // print stats
    env.PrintTimeStat(stdout);

    // write out solutions
    std::string filename("sol.txt");
    writeSolution(env, solution_stateIDs, filename.c_str());
    delete planner;
}

int main(int argc, char *argv[])
{
    planxythetalat(argv[1], argv[2]);
}
```

Listing 3: Source Code of the `xytheta.cpp` C++ file

References

[1] `sw_urdf_exporter` Package Reference
`http://wiki.ros.org/sw_urdf_exporter`

[2] SBPL Reference
`http://sbpl.net/`

[3] `Gazebo` Robot Simulator
`http://gazebosim.org/`