24-783 : Advanced Engineering Computation Report

rchavali@andrew.cmu.edu

Title: 3d visualization of a multi agent 2d path planning algorithm.

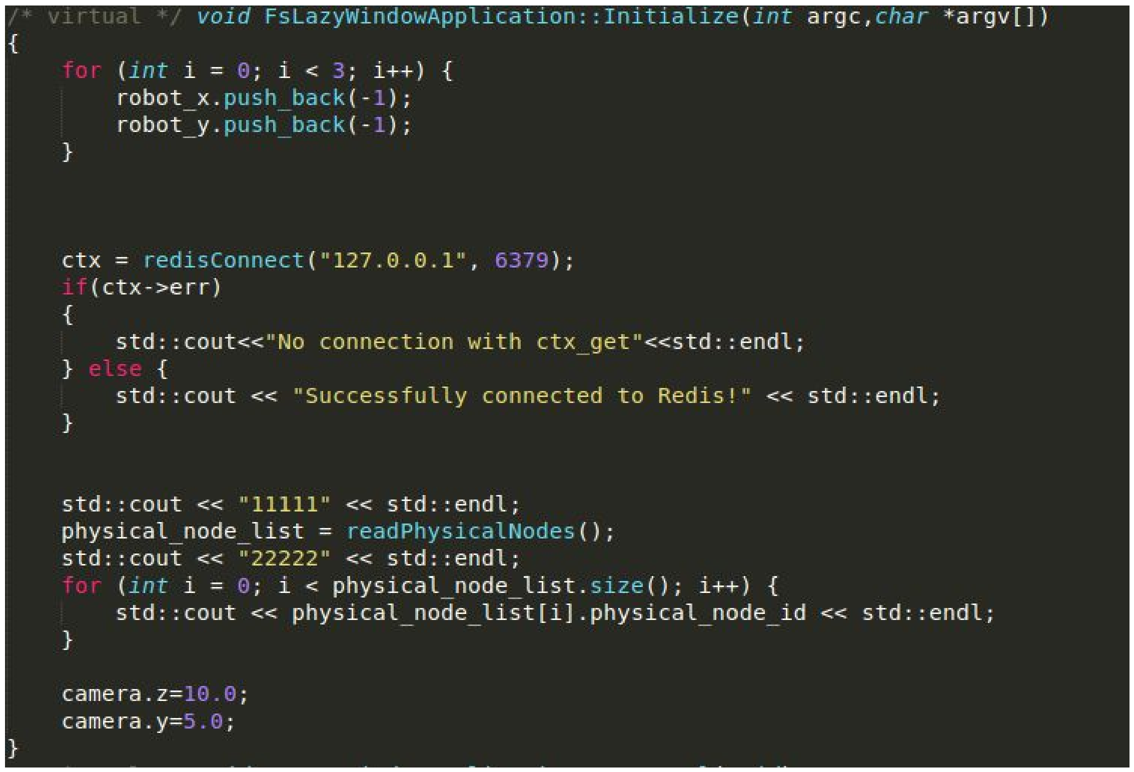
Area of prime contribution in code : C++ visualization and integrating the path planning part with OpenGL

Area of prime contribution in theoretical knowledge: Path planning algorithm

After assisting the team in coming up with a simple algorithm, we spent countless hours trying to debug the path planning algorithm. Once the alpha version of the path planning code started working, I started looking at ways in which the path planning might be integrated with the C++ code to get a 3d visualization. This report goes over all the different targets achieved and the work that was yet to be completed (by gautham) in the preliminary codes that were attached.

The targets that were achieved are:

1. Realizing the use of redis.
    a. Redis is an open source, in-memory data structure store, used as a database, cache and message broker. I decided to use this as a link between the path planning and 3d visualization algorithm. The C client for this redis database is **hiredis**.

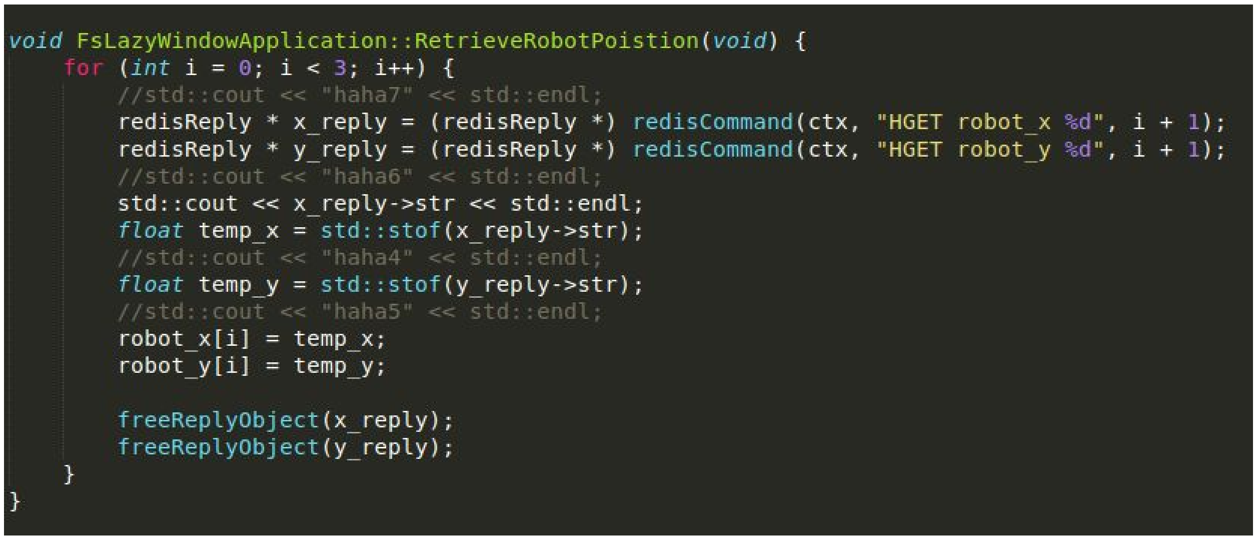


```
/* virtual */ void FsLazyWindowApplication::Initialize(int argc,char *argv[])
{
    for (int i = 0; i < 3; i++) {
        robot_x.push_back(-1);
        robot_y.push_back(-1);
    }



    ctx = redisConnect("127.0.0.1", 6379);
    if(ctx->err)
    {
        std::cout<<"No connection with ctx_get"<<std::endl;
    } else {
        std::cout << "Successfully connected to Redis!" << std::endl;
    }


    std::cout << "11111" << std::endl;
    physical_node_list = readPhysicalNodes();
    std::cout << "22222" << std::endl;
    for (int i = 0; i < physical_node_list.size(); i++) {
        std::cout << physical_node_list[i].physical_node_id << std::endl;
    }

    camera.z=10.0;
    camera.y=5.0;
}
```

This is done in the initialization.

2. Successfully reading the coordinates of the robot and the map from redis:
   a. void FsLazyWindowApplication::RetrieveRobotPoistion(void)

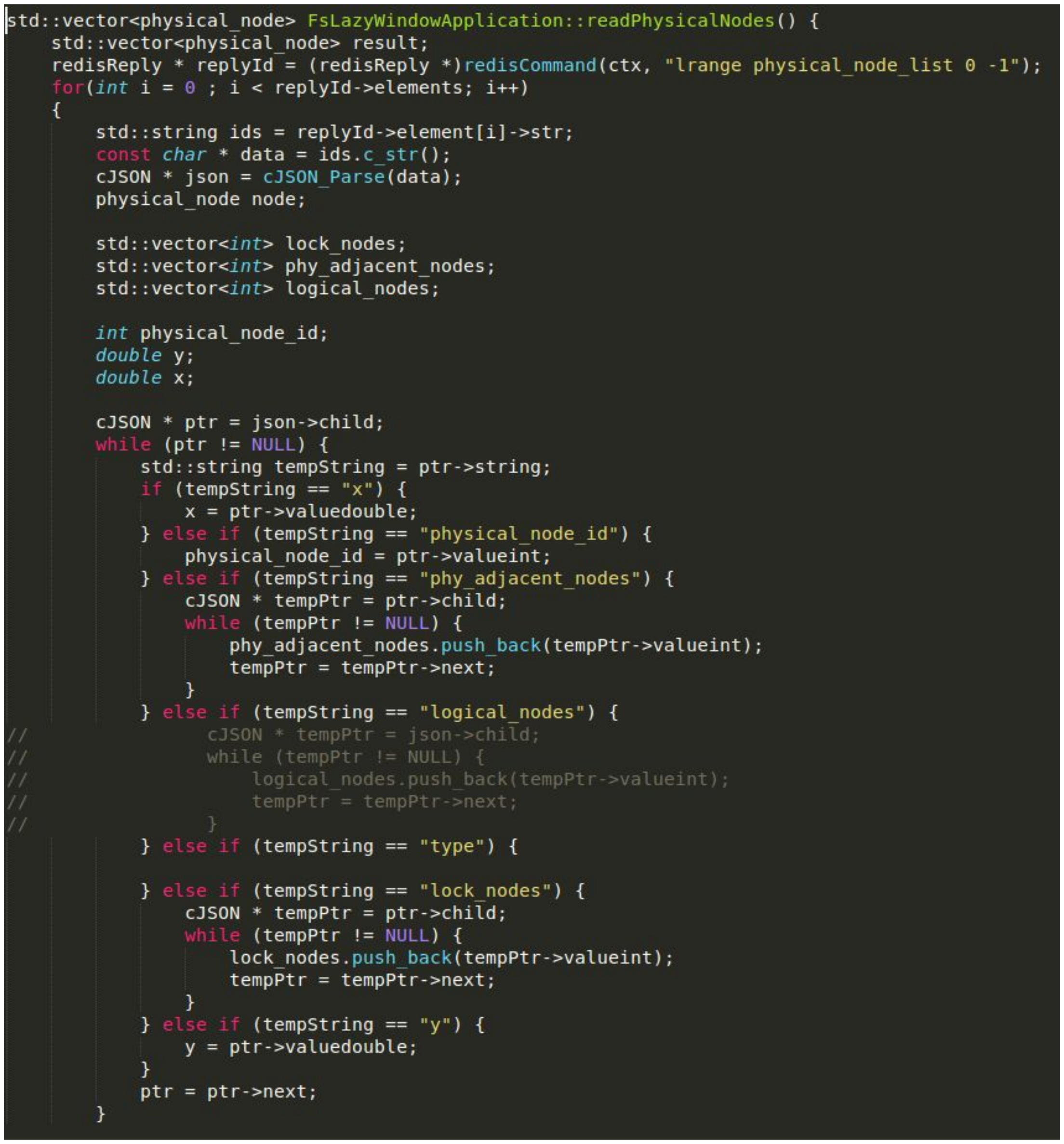
      This function retrieves the (x,y) position of three different robots (ie, {(x1,y1), (x2,y2), (x3,y3)}) and stores it such that it can be used by the OpenGL functions to draw and visualize its movements.



```
void FsLazyWindowApplication::RetrieveRobotPoistion(void) {
    for (int i = 0; i < 3; i++) {
        //std::cout << "haha7" << std::endl;
        redisReply * x_reply = (redisReply *) redisCommand(ctx, "HGET robot_x %d", i + 1);
        redisReply * y_reply = (redisReply *) redisCommand(ctx, "HGET robot_y %d", i + 1);
        //std::cout << "haha6" << std::endl;
        std::cout << x_reply->str << std::endl;
        float temp_x = std::stof(x_reply->str);
        //std::cout << "haha4" << std::endl;
        float temp_y = std::stof(y_reply->str);
        //std::cout << "haha5" << std::endl;
        robot_x[i] = temp_x;
        robot_y[i] = temp_y;

        freeReplyObject(x_reply);
        freeReplyObject(y_reply);
    }
}
```

b. std::vector<physical_node> FsLazyWindowApplication::readPhysicalNodes()

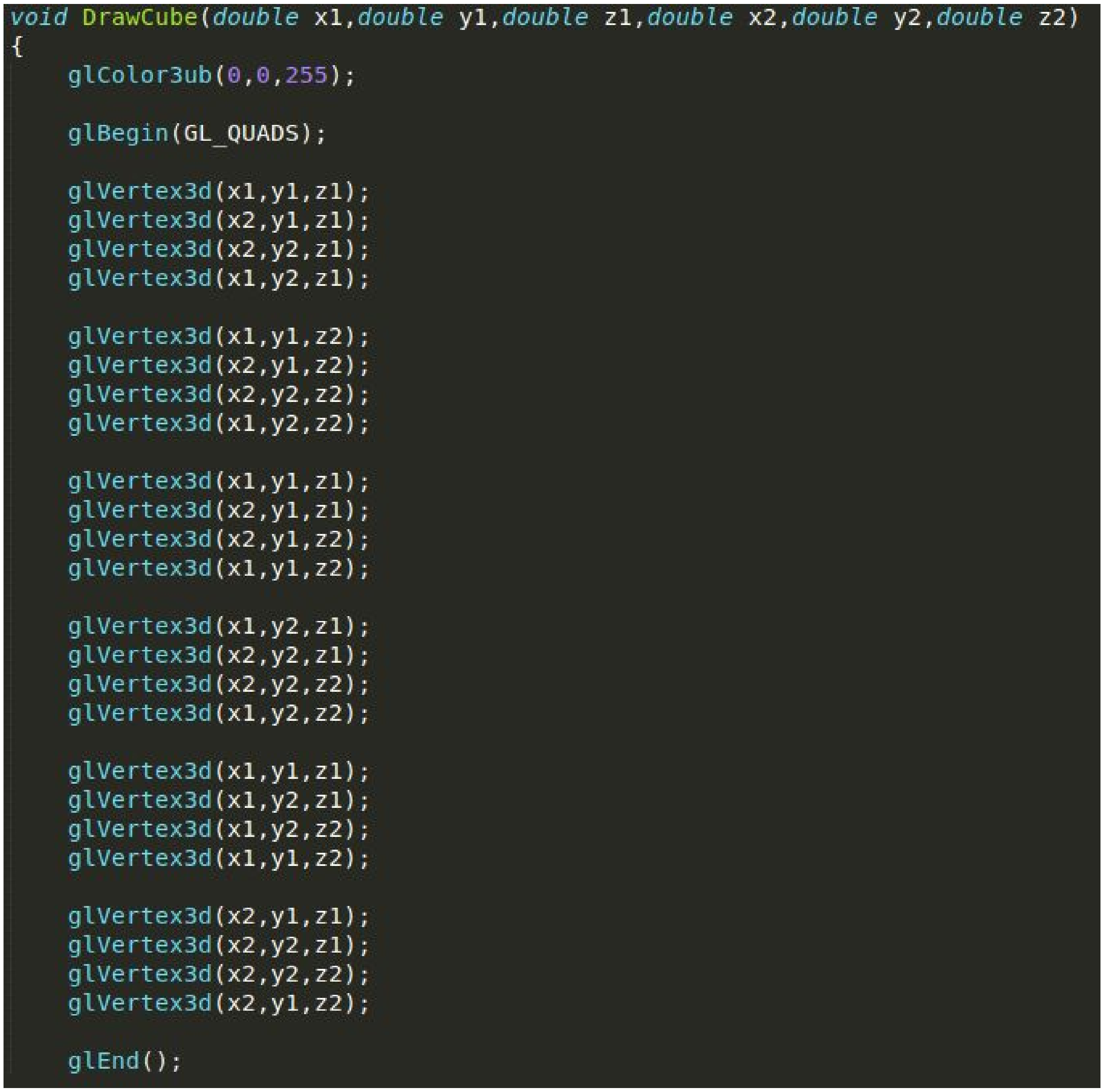
This function reads the map that was used by the robots for the path planning task. The map is constructed as a set of nodes placed equidistant from each other. The physical_node is a structure that contains the (x,y) coordinates of all the nodes in the map.



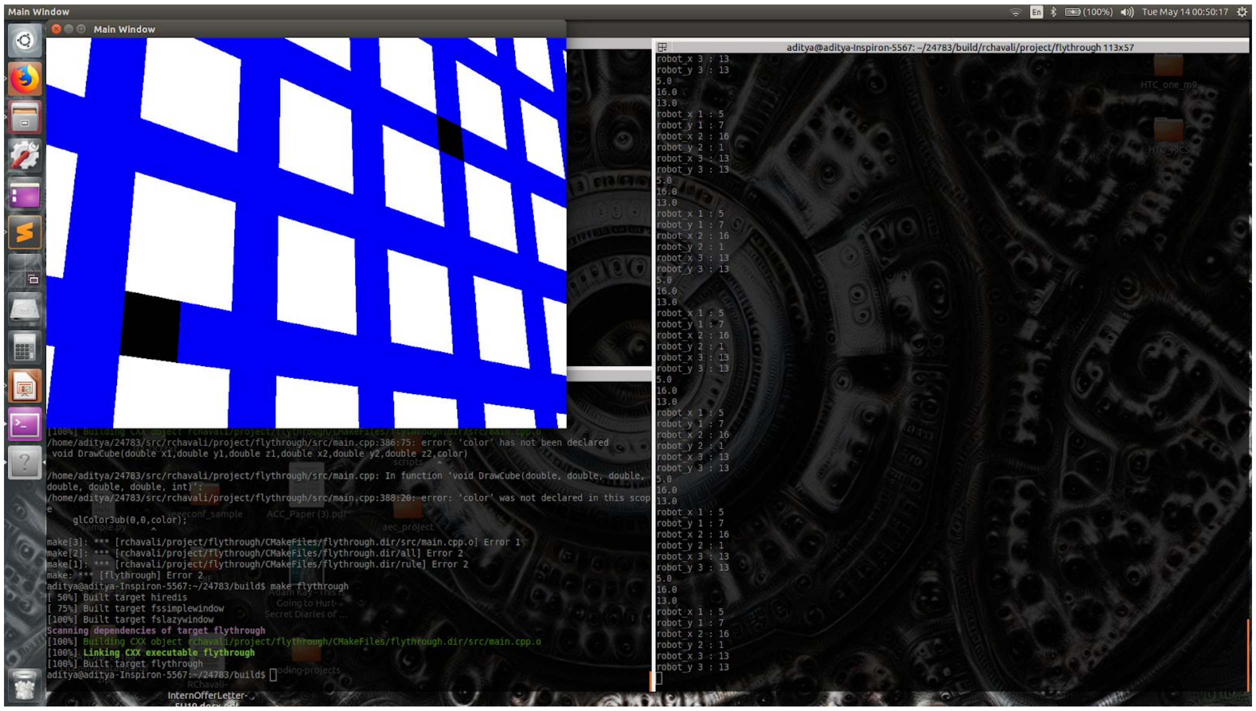
```cpp
std::vector<physical_node> FsLazyWindowApplication::readPhysicalNodes() {
    std::vector<physical_node> result;
    redisReply * replyId = (redisReply *)redisCommand(ctx, "lrange physical_node_list 0 -1");
    for(int i = 0 ; i < replyId->elements; i++)
    {
        std::string ids = replyId->element[i]->str;
        const char * data = ids.c_str();
        cJSON * json = cJSON_Parse(data);
        physical_node node;

        std::vector<int> lock_nodes;
        std::vector<int> phy_adjacent_nodes;
        std::vector<int> logical_nodes;

        int physical_node_id;
        double y;
        double x;

        cJSON * ptr = json->child;
        while (ptr != NULL) {
            std::string tempString = ptr->string;
            if (tempString == "x") {
                x = ptr->valuedouble;
            } else if (tempString == "physical_node_id") {
                physical_node_id = ptr->valueint;
            } else if (tempString == "phy_adjacent_nodes") {
                cJSON * tempPtr = ptr->child;
                while (tempPtr != NULL) {
                    phy_adjacent_nodes.push_back(tempPtr->valueint);
                    tempPtr = tempPtr->next;
                }
            } else if (tempString == "logical_nodes") {
//                  cJSON * tempPtr = json->child;
//                  while (tempPtr != NULL) {
//                      logical_nodes.push_back(tempPtr->valueint);
//                      tempPtr = tempPtr->next;
//                  }
            } else if (tempString == "type") {

            } else if (tempString == "lock_nodes") {
                cJSON * tempPtr = ptr->child;
                while (tempPtr != NULL) {
                    lock_nodes.push_back(tempPtr->valueint);
                    tempPtr = tempPtr->next;
                }
            } else if (tempString == "y") {
                y = ptr->valuedouble;
            }
            ptr = ptr->next;
        }
```

3. Flythrough was used as a base code in order to complete this task.

4. The part that was left to do was to draw 3d rectangles at each of the x,y coordinates of the nodes in the map. This is when the incident with my family took place and I had to immediately leave. The following is the preliminary code and results of the visualization:



```
void DrawCube(double x1,double y1,double z1,double x2,double y2,double z2)
{
    glColor3ub(0,0,255);

    glBegin(GL_QUADS);

    glVertex3d(x1,y1,z1);
    glVertex3d(x2,y1,z1);
    glVertex3d(x2,y2,z1);
    glVertex3d(x1,y2,z1);

    glVertex3d(x1,y1,z2);
    glVertex3d(x2,y1,z2);
    glVertex3d(x2,y2,z2);
    glVertex3d(x1,y2,z2);

    glVertex3d(x1,y1,z1);
    glVertex3d(x2,y1,z1);
    glVertex3d(x2,y1,z2);
    glVertex3d(x1,y1,z2);

    glVertex3d(x1,y2,z1);
    glVertex3d(x2,y2,z1);
    glVertex3d(x2,y2,z2);
    glVertex3d(x1,y2,z2);

    glVertex3d(x1,y1,z1);
    glVertex3d(x1,y2,z1);
    glVertex3d(x1,y2,z2);
    glVertex3d(x1,y1,z2);

    glVertex3d(x2,y1,z1);
    glVertex3d(x2,y2,z1);
    glVertex3d(x2,y2,z2);
    glVertex3d(x2,y1,z2);

    glEnd();
```


Main Window
Tue May 14 00:50:17
Main Window
aditya@aditya-Inspiron-5567: ~/24783/build/rchavali/project/flythrough 113x57

```
[100%] Building CXX object rchavali/project/flythrough/CMakeFiles/flythrough.dir/src/main.cpp.o
/home/aditya/24783/src/rchavali/project/flythrough/src/main.cpp:386:75: error: 'color' has not been declared
 void DrawCube(double x1,double y1,double z1,double x2,double y2,double z2,color)

/home/aditya/24783/src/rchavali/project/flythrough/src/main.cpp: In function 'void DrawCube(double, double, double,
double, double, double, int)':
/home/aditya/24783/src/rchavali/project/flythrough/src/main.cpp:388:20: error: 'color' was not declared in this scop
e
     glColor3ub(0,0,color);
                    ^
make[3]: *** [rchavali/project/flythrough/CMakeFiles/flythrough.dir/src/main.cpp.o] Error 1
make[2]: *** [rchavali/project/flythrough/CMakeFiles/flythrough.dir/all] Error 2
make[1]: *** [rchavali/project/flythrough/CMakeFiles/flythrough.dir/rule] Error 2
make: *** [flythrough] Error 2
aditya@aditya-Inspiron-5567:~/24783/build$ make flythrough
[ 50%] Built target hiredis
[ 75%] Built target fssimplewindow
[100%] Built target fslazywindow
Scanning dependencies of target flythrough
[100%] Building CXX object rchavali/project/flythrough/CMakeFiles/flythrough.dir/src/main.cpp.o
[100%] Linking CXX executable flythrough
[100%] Built target flythrough
aditya@aditya-Inspiron-5567:~/24783/build$ 
```

```
robot_x 3 : 13
robot_y 3 : 13
5.0
16.0
13.0
robot_x 1 : 5
robot_y 1 : 7
robot_x 2 : 16
robot_y 2 : 1
robot_x 3 : 13
robot_y 3 : 13
5.0
16.0
13.0
robot_x 1 : 5
robot_y 1 : 7
robot_x 2 : 16
robot_y 2 : 1
robot_x 3 : 13
robot_y 3 : 13
5.0
16.0
13.0
robot_x 1 : 5
robot_y 1 : 7
robot_x 2 : 16
robot_y 2 : 1
robot_x 3 : 13
robot_y 3 : 13
5.0
16.0
13.0
robot_x 1 : 5
robot_y 1 : 7
robot_x 2 : 16
robot_y 2 : 1
robot_x 3 : 13
robot_y 3 : 13
5.0
16.0
13.0
robot_x 1 : 5
robot_y 1 : 7
robot_x 2 : 16
robot_y 2 : 1
robot_x 3 : 13
robot_y 3 : 13
5.0
16.0
13.0
robot_x 1 : 5
robot_y 1 : 7
robot_x 2 : 16
robot_y 2 : 1
robot_x 3 : 13
robot_y 3 : 13
```

In the above picture, the map is shown in blue and the robots are drawn in black. However, the setback in this was that the map was not a 3d fly through as we expected it to be. Instead it was a giant wall.

Gautham said that he would continue and work on this, but since he was more comfortable with python, and he was giving a presentation, I helped him code this hiredis connection and visualization on python. I did so from a remote place getting access to his desktop using teamviewer.

Conclusion:

The only part remaining in this preliminary code was drawing the rectangles in the desired position. But for reasons beyond my control, this code was never used in the final version of the code.
I am attaching the codes for your reference.

Thank you.