```
1
2  #include "ros/ros.h"
3  #include "std_msgs/String.h"
4  #include "std_msgs/Float32.h"
5  #include "std_msgs/Int32.h"
6  #include <thread>
7  #include <unistd.h>
8  #include <iostream>
9  #include <chrono>
10 #include <signal.h>
11 #include <stdlib.h>
12 #include <stdio.h>
13 #include "opencv_node/vision_msg.h"
14 #include <geometry_msgs/PoseWithCovarianceStamped.h>
15 #include "opencv_node/object.h"
16 #include <move_base_msgs/MoveBaseAction.h>
17 #include <actionlib/client/simple_action_client.h>
18 #include <tf/transform_broadcaster.h>
19
20 //#include "sensor_msgs/Imu.h"
21 //#include "Arduino-Serial/ArduinoSerial.h"
22 //Color Indices = red(0), yellow(1), blue(2), green(3)
23 using namespace std;
24
25 //serialPort arduino("/dev/serial/by-id/usb-1a86_USB2.0-Serial-if00-port0");
26
27
28 typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
   MoveBaseClient;
29 std_msgs::String msg;
30 int rightSpeed=0,leftSpeed=0;
31 double closestBlockX = 0.0;//add this to y for map placement
32 double closestBlockY = 0.0;//add this to x for map placement
33 int numberBlocks = 0; //number of blocks seen
34 double desiredColor = 0.0;
35 string colorSelect = "0"; //recieved startColor
36 int colorChoose = 0;
37 int goalMet = 0;
38 int octetNum = 0;
39 double dummyRobotX = 0.0;
40 double dummyRobotY = 0.0;
41 int startMatch = 0;
42 int loopNum = 0;
43 double initialPose[2] = {0.0,0.0};
44 int moveBaseTest = 0;
45
46 void colorSelected(const std_msgs::Float32ConstPtr &msg){
47   colorChoose = int(msg->data);
48 }
49 //void matchStarted(const std_msgs::Int32ConstPtr &msg){
50 //  startMatch = int(msg->data);
51 //}
52
53 double objDistance(const opencv_node::object& obj) {
54   return sqrt(pow(obj.x_position, 2) + pow(obj.y_position, 2));
55 }
56
57 void visionCallback(const opencv_node::vision_msg::ConstPtr &msg)
58 {
59   ROS_INFO("Main>>>Number of Objects: %d", msg->objects.size());
```

```
60   numberBlocks = msg->objects.size();
61   int desiredColor = 0;
62   double minDistance = 0.0;
63   int currentMin = -1;
64   for (int i = 0; i < msg->objects.size(); ++i)
65   {
66     const opencv_node::object &prop = msg->objects[i];
67     ROS_INFO_STREAM("Position: " << prop.x_position << "," << prop.y_position << "
Color:" << prop.color_index << " Object Type:" << prop.object_type);
68     if(prop.color_index == desiredColor && (currentMin == -1 || objDistance(prop) <
minDistance)) {
69       currentMin = i;
70       minDistance = objDistance(prop);
71     }
72   }
73   //auto closest = min_element(msg->objects.begin(),msg->objects.end(),[](const
opencv_node::object &first,const opencv_node::object &second){
74     //return objDistance(first) < objDistance(second);
75   //});
76   if(currentMin != -1) {
77     closestBlockX = msg->objects[currentMin].x_position;
78     closestBlockY = msg->objects[currentMin].y_position;
79     desiredColor = msg->objects[currentMin].color_index;//not used yet
80     ROS_INFO_STREAM("Selected Object >>> Position: " << msg-
>objects[currentMin].x_position << "," << msg->objects[currentMin].y_position << "
Color:" << msg->objects[currentMin].color_index << " Object Type:" << msg-
>objects[currentMin].object_type);
81   }
82
83
84 }
85
86 void moveFwdOneMeter(){
87   //MOVE BASE CODE//
88   int counter = 1;
89         MoveBaseClient ac("/move_base", true); //Tell the client we want to spin a
thread by default
90          while(!ac.waitForServer(ros::Duration(5.0))){
91                 ROS_INFO("Waiting for the move_base action server to come up");
92         }
93
94         move_base_msgs::MoveBaseGoal moveFwd;
95
96   moveFwd.target_pose.header.frame_id = "base_footprint";
97         moveFwd.target_pose.header.stamp = ros::Time::now();
98
99   //if(counter){
100           moveFwd.target_pose.pose.position.x = 0.5; //move 1 meter forward
101     moveFwd.target_pose.pose.orientation = tf::createQuaternionMsgFromYaw(0.0);
102     counter = 0;
103   //}
104   //else{
105   /*  counter++;
106     moveFwd.target_pose.pose.position.x = 0.5;
107     moveFwd.target_pose.pose.orientation = tf::createQuaternionMsgFromYaw(-90.0);
108   }*/
109         ROS_INFO("Sending goal");
110   ac.sendGoal(moveFwd);
111
112   //ac.waitForResult();
```

```
113
114   //if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
115   //  ROS_INFO("WOOP WOOP YOU DID IT");
116   //else
117   //  ROS_INFO("You screwed up boi");
118         ////////////////
119
120 }
121
122 //Note: frame is typicall "map" or "base_footprint"
123 bool moveToGoal(double xGoal, double yGoal, string frame){
124    //define a client for to send goal requests to the move_base server through a
    SimpleActionClient
125    actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> ac("/move_base",
    true);
126
127    //wait for the action server to come up
128    while(!ac.waitForServer(ros::Duration(5.0))){
129       ROS_INFO("Waiting for the move_base action server to come up");
130    }
131
132    move_base_msgs::MoveBaseGoal goal;
133
134    //set up the frame parameters
135    goal.target_pose.header.frame_id = frame;
136    goal.target_pose.header.stamp = ros::Time::now();
137
138    /* moving towards the goal*/
139
140    goal.target_pose.pose.position.x =  xGoal;
141    goal.target_pose.pose.position.y =  yGoal;
142    goal.target_pose.pose.position.z =  0.0;
143    goal.target_pose.pose.orientation.x = 0.0;
144    goal.target_pose.pose.orientation.y = 0.0;
145    goal.target_pose.pose.orientation.z = 0.0;
146    goal.target_pose.pose.orientation.w = 1.0;
147
148    ROS_INFO("Sending goal location ...");
149    ac.sendGoal(goal);
150    ac.waitForResult();
151
152    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
153       ROS_INFO("You have reached the destination");
154       goalMet = 1;
155       return true;
156    }
157    else{
158       ROS_INFO("The robot failed to reach the destination");
159       goalMet = 0;
160       return false;
161    }
162
163 }
164
165 // void arduinoCallback(const std_msgs::String& msg) {
166 //  ROS_INFO_STREAM(msg << '\n');
167 // }
168
169 /*void imuCallback(const sensor_msgs::Imu::ConstPtr &msg)
170 {
```

```
171   float gyro_x = msg->angular_velocity.x;
172   float gyro_y = msg->angular_velocity.y;
173   float gyro_z = msg->angular_velocity.z;
174   float orientation_x = msg->orientation.x;
175   float orientation_y = msg->orientation.y;
176   float orientation_z = msg->orientation.z;
177   float orientation_w = msg->orientation.w;
178   //ROS_INFO("Main>>>Angular Velocity: x(%f),y(%f),z(%f)", gyro_x,gyro_y,gyro_z);
179   ROS_INFO("Main>>>Orientation: x(%f),y(%f),z(%f)", orientation_x, orientation_y,
    orientation_z);
180 }*/
181
182 bool isWaitingForAction = false;
183 int blockCount = 0;
184 enum class State {
185   initialMovement,
186   findBlocks,
187   findCorner
188 };
189
190
191
192
193 int main(int argc, char **argv)
194 {
195
196   ros::init(argc, argv, "main");
197   ros::NodeHandle n;
198   ros::Subscriber sub = n.subscribe("vision_info", 1000, visionCallback);
199
200
201
202  // initPose = n.advertise<geometry_msgs::PoseWithCovarianceStamped>
    ("initialpose",1,true);
203
204   cout << "\033[1;34m--------------------------------------------------------------
    ----\033[0m" << endl;
205   cout << "\033[1;34m   .:: ::    .:::::::          .:         .:::::::          .::
     \033[0m" << endl;
206   cout << "\033[1;34m .::    .::  .::    .::       .: ::       .::    .::     .::
    .:: \033[0m" << endl;
207   cout << "\033[1;34m  .::        .::    .::      .:  .::      .::    .::    .::
     \033[0m" << endl;
208   cout << "\033[1;34m    .::      .::::::::       .::   .::     .: .::        .::
     \033[0m" << endl;
209   cout << "\033[1;34m       .::   .::           .:::::: .::    .::  .::      .::
     \033[0m" << endl;
210   cout << "\033[1;34m .::    .::  .::          .::       .::   .::    .::     .::
    .:: \033[0m" << endl;
211   cout << "\033[1;34m   .:: ::    .::         .::         .::  .::      .::
    .::::   \033[0m" << endl;
212   cout << "\033[1;34m--------------------------------------------------------------
    ----\033[0m" << endl;
213   cout << "\033[1;34m|        Student Projects and Research Committee IEEE 2019
     |\033[0m" << endl;
214   cout << "\033[1;34m ------------------------------------------------------------
    ----\033[0m" << endl;
215   sleep(1);
216   //ros::Subscriber sub2 = n.subscribe("sensor_msgs/Imu", 1000, imuCallback);
217
```

```
218   ros::Publisher colorSelectPub = n.advertise<std_msgs::Int32>
("colorSelect",1);//this publishes data to vision shit
219   ros::Publisher gate_cmd = n.advertise<std_msgs::Int32>("gate_cmd",1);
220   ros::Publisher flag_cmd = n.advertise<std_msgs::Int32>("flag_cmd",1);
221   ros::Publisher color_Want = n.advertise<std_msgs::Float32>("color_want",1);
222
223   ros::Subscriber startColorSub = n.subscribe<std_msgs::Float32>("start_color", 1,
colorSelected);
224   //ros::Subscriber startMatchSub = n.subscribe<std_msgs::Int32>("startMatchFunc",
1, matchStarted);
225   ros::Rate loop_rate(40);  //1 Hz
226
227
228   //geometry_msgs::PoseWithCovarianceStamped ip;
229   //ip.header.frame_id = "map";
230   ros::Time current_time = ros::Time::now();
231   /*ip.header.stamp = current_time;
232   ip.pose.pose.position.x = 0.1143;
233   ip.pose.pose.position.y = 0.1143;
234   ip.pose.pose.orientation.z = 0;
235   ip.pose.covariance[0] = 1e-3;
236   ip.pose.covariance[7] = 1e-3;
237   ip.pose.covariance[35] = 1e-3;
238   initPose.publish(ip);
239   */
240   int count = 0;
241   bool done = 0;
242   double myGoalX[8] = {-0.35,-0.5,-0.35,0,0.35,0.5,0.35,0};
243   double myGoalY[8] = {0.35,0,-0.35,-0.5,-0.35,0,0.35,0.5};//set these according to
the new map locations
244  // double myGoalZ[8] = {-2.36,-1.57,0.785,0.0,0.785,1.571,2.36,0.5};//set these
according to the new map locations
245
246   while(ros::ok()) {
247
248     msg.data = std::string("Hello ");
249     msg.data += std::to_string(count);
250     //this sets the selected
251
252   //  if(startMatch > 0){
253     //this tests the octet and debris goal setting
254       //system("roslaunch /home/ubuntu/ieee-2019-electrical-software/launches
amcl_diff.launch");
255         ROS_INFO("WE GOT HERE");
256       while(count<15){
257         moveToGoal(0.5,0, "base_footprint");
258         moveToGoal(-0.5,-0.5, "map");
259         moveToGoal(-0.5,0, "map");
260         moveToGoal(0.5,-0.5, "map");
261         moveToGoal(0.5, 0, "map");
262         moveToGoal(0.5,0.5, "map");
263         moveToGoal(-0.5, 0.5, "map");
264         //ros::spinOnce();
265         loop_rate.sleep();
266         ++count;
267       }
268     //}
269 ros::spinOnce();
270 }
271
```

```
272   return 0;
273
274 }
275
276
```