src/drive/src/TB3Drive.cpp

```cpp
#include "TB3Drive.h"

// Implementation file for class TB3Drive
// Functions :
//            - Constructor
//            - Destructor
//            - Call back function sub to CLidar topic
//            - Call back function sub to CPose msg
//            - Setting linear and angular velocities of bot
//            - Control loop function
//            - Main NODE

//---Constructor
TB3Drive::TB3Drive(): nh_priv_("~")
{
  //Init gazebo ros turtlebot3 node
  ROS_INFO("Turtlebot3 Drive node initalised");

    // initialize ROS parameter
  std::string cmd_vel_topic_name = nh_.param<std::string>("cmd_vel_topic_name",
"");

  forwardTarget = 0.3;
  sideTarget = 0.25;

  forwardTargetTurn = 0.3;

  // Maximum values for preventing overshoot
  maxTurnVel = 1.0;
  maxForwardVel = 0.15;
  minForwardVel = 0.0;

  // Proportional gains
  forwardKp = 0.5;
  turnKp = 6.0;

  // Default turn left turn flag
  leftTurnFlag = 3;

  // Set default values to 0
  tb3Pose = 0.0;
  prevTB3pose = 0.0;

  angularVel = 0.0;
  linearVel = 0.0;

  // Populate Vector with default 0.0 lidar scan values
  for (int i = 0; i < 3; i++)
  {
    lidarData.push_back(0.0);
  }

  // initialize publishers
  cmd_vel_pub_   = nh_.advertise<geometry_msgs::Twist>(cmd_vel_topic_name,
1000);

```

```
55    // initialize subscribers
56    cLidarSub = nh_.subscribe("LIDAR", 1000, &TB3Drive::cLidarMsgCallBack, this);
57    cBotSub = nh_.subscribe("POSE", 1000, &TB3Drive::cPoseMsgCallBack, this);
58
59    ROS_ASSERT(true);
60  }
61
62  //---Destructor
63  TB3Drive::~TB3Drive()
64  {
65    lidarData.clear();
66    updatecommandVelocity(0.0, 0.0);
67    ros::shutdown();
68  }
69
70  //---Call back function sub to CLidar topic
71  void TB3Drive::cLidarMsgCallBack(const std_msgs::Float64MultiArray::ConstPtr &
    msg)
72  {
73    lidarData.clear();
74    for (int i = 0; i < msg->data.size(); i ++){
75      lidarData.push_back(msg->data[i]);
76    }
77  }
78
79  //---Call back function sub to CPose msg
80  void TB3Drive::cPoseMsgCallBack(const std_msgs::Float64::ConstPtr &msg)
81  {
82    tb3Pose = msg->data;
83  }
84
85  //---Setting linear and angular velocities of bot
86  void TB3Drive::updatecommandVelocity(double linear, double angular)
87  {
88    geometry_msgs::Twist cmd_vel;
89
90    cmd_vel.linear.x  = linear;
91    cmd_vel.angular.z = angular;
92
93    cmd_vel_pub_.publish(cmd_vel);
94  }
95
96  //---Control loop function
97  // Function check flags for states transitions and compute linear and angular
    vel
98  // using proportional gains
99  bool TB3Drive::controlLoop()
100 {
101   // check for left turn flag
102   if ((lidarData[CENTER] <= forwardTarget)&&(leftTurnFlag==STRAIGHT))
103   {
104     leftTurnFlag = LEFT_TURN;  // check for left turn flag
105   }
106   else if((lidarData[CENTER] != 0)&&(leftTurnFlag==DEFAULT_STATE))
107   {
108     leftTurnFlag= STRAIGHT;
109   }
110
111   // angular velocity
```

```
112   angularVel= turnKp*(sideTarget-lidarData[RIGHT]);
113
114   if(angularVel > maxTurnVel)
115   {
116     angularVel = maxTurnVel;
117   }
118   else if(angularVel < (-1.0)*maxTurnVel)
119   {
120     angularVel= (-1.0)*maxTurnVel;
121   }
122
123
124   linearVel = maxForwardVel;
125
126   if(linearVel  > maxForwardVel)
127   {
128     linearVel = maxForwardVel;
129   }
130   else if(linearVel  <= minForwardVel)
131   {
132     linearVel  = minForwardVel;
133   }
134
135   if ( leftTurnFlag>= LEFT_TURN)  // if left turn flag set, go left turn,
    otherwise do normal right wall fellower
136   {
137     linearVel = STOP_FOWARD_V;
138     angularVel= maxTurnVel;
139
140     if((lidarData[CENTER] >= forwardTargetTurn)&&(leftTurnFlag== LEFT_TURN))//
    if left turn 90 degree, go for normal right wall fellower, set flag to 0
141     {
142       leftTurnFlag = CORNER_TURN;
143     }
144     else if((lidarData[RIGHT] >= sideTarget)&&(leftTurnFlag==CORNER_TURN)) //
    if left turn 90 degree, go for normal right wall fellower, set flag to 0
145     {
146       leftTurnFlag = STRAIGHT;
147     }
148   }
149
150   updatecommandVelocity(linearVel, angularVel);
151
152   return true;
153 }
154
155 //-------------------------------------------------------------
156 // TB3Drive NODE
157 int main(int argc, char* argv[])
158 {
159   ros::init(argc, argv, "Drive_Node");
160   TB3Drive drive;
161
162   ros::Rate loop_rate(500);
163
164   while (ros::ok())
165   {
166     bool b  = drive.controlLoop();
167
```

```
168       // process callback for this node
169       ros::spinOnce();
170       loop_rate.sleep();
171     }
172     return 0;
173   }
```