src/drive/src/TB3Drive.cpp

```cpp
#include "TB3Drive.h"

// Implementation file for class CLidar
// 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", "");

  // Initliase variables used for checking distances ( unit is meters)
  escapeRange        = 30.0 * DEG2RAD;
  checkForwardDist   = 0.3;
  checkSideDist      = 0.5;

  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 = 0.0;

  // Default turn left turn flag
  leftTurnFlag = TB3_LEFT_TURN;

  // 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);

  // initialize subscribers
  cLidarSub = nh_.subscribe("LIDAR", 1000, &TB3Drive::cLidarMsgCallBack, this);
  cBotSub = nh_.subscribe("POSE", 1000, &TB3Drive::cPoseMsgCallBack, this);

  ROS_ASSERT(true);
}

//---Destructor
TB3Drive::~TB3Drive()
{
  lidarData.clear();
  updatecommandVelocity(0.0, 0.0);
  ros::shutdown();
}

//---Call back function sub to CLidar topic
void TB3Drive::cLidarMsgCallBack(const std_msgs::Float64MultiArray::ConstPtr & msg)
{
  lidarData.clear();
  for (int i = 0; i < msg->data.size(); i ++){
    lidarData.push_back(msg->data[i]);
  }

  ROS_INFO("lefT: %f | MID: %f | Right: %f ",lidarData[0],lidarData[1], lidarData[2]);
}

//---Call back function sub to CPose msg
void TB3Drive::cPoseMsgCallBack(const std_msgs::Float64::ConstPtr &msg)
{
  tb3Pose = msg->data;
}

//---Setting linear and angular velocities of bot
void TB3Drive::updatecommandVelocity(double linear, double angular)
{
  geometry_msgs::Twist cmd_vel;

  cmd_vel.linear.x  = linear;
  cmd_vel.angular.z = angular;

  cmd_vel_pub_.publish(cmd_vel);
}

//---Control loop function
// Function check flags for states transitions and compute linear and angular vel
// using proportional gains
bool TB3Drive::controlLoop()
{
  //CHECK FLAGS
  // check for left turn flag
  if ((lidarData[CENTER] <= forwardTarget)&&(leftTurnFlag==GET_TB3_DIRECTION))
```

```cpp
 {
  leftTurnFlag = TB3_DRIVE_FORWARD ;  // check for left turn flag
 }
 else if((lidarData[CENTER] != 0)&&(leftTurnFlag==TB3_LEFT_TURN ))
 {
  leftTurnFlag = GET_TB3_DIRECTION;
 }

 // COMPUTE ANGULAR AND LINEAR VELOCITIES
 angularVel = turnKp*(sideTarget-lidarData[RIGHT]);

 if(angularVel  > maxTurnVel)
 {
  angularVel  = maxTurnVel;
 }
 else if(angularVel < (-1.0)*maxTurnVel)
 {
  angularVel  = maxTurnVel * (-1.0);
 }

 linearVel = maxForwardVel;

 if(linearVel > maxForwardVel)
 {
  linearVel = maxForwardVel;
 }
 else if(linearVel <= maxForwardVel)
 {
  linearVel = maxForwardVel;
 }

 // if left turn flag set, go left turn, otherwise do normal right wall follower
 if ( leftTurnFlag >= TB3_DRIVE_FORWARD)
 {
  linearVel = 0.0;
  angularVel= maxTurnVel;

  if((lidarData[CENTER] >= forwardTargetTurn)&&(leftTurnFlag == TB3_DRIVE_FORWARD))
  {
   // if left turn 90 degree, go for normal right wall fellower, set flag to 0
   leftTurnFlag = TB3_RIGHT_TURN;
  }
  else if((lidarData[RIGHT] >= sideTarget)&&(leftTurnFlag==TB3_RIGHT_TURN  ))
  {
   // if left turn 90 degree, go for normal right wall fellower, set flag to 0
   leftTurnFlag = GET_TB3_DIRECTION;
  }
 }

 // publish new velocities to Twist
 updatecommandVelocity(linearVel, angularVel);

 return true;
}

//---------------------------------------------------------------
```

```cpp
// TB3Drive NODE
int main(int argc, char* argv[])
{
  ros::init(argc, argv, "Drive_Node");
  TB3Drive drive;

  ros::Rate loop_rate(125);

  while (ros::ok())
  {
    bool b  = drive.controlLoop();

    // process callback for this node
    ros::spinOnce();
    loop_rate.sleep();
  }
  return 0;
}

```