MTRX3760 Project 1 Report

SID	Tutorial
460311650	LAB 03
460418627	LAB 03
500554025	LAB 03
500562776	LAB 03

Assignment Due Date: 6 October 2023

Contents

Design

ROS Node Design Diagram

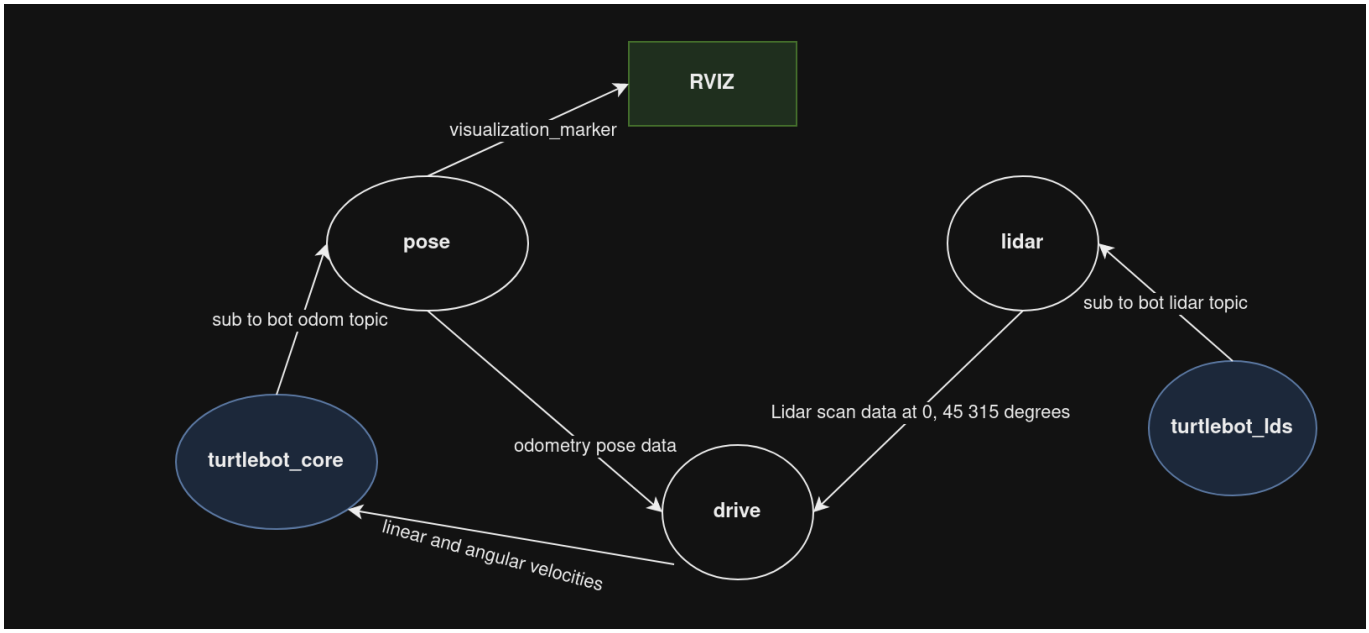


Figure 1: Ros node design of project

- **turtlebot3_core**: Existing node upon turtlebot bring up. This node subscribes to /cmd_vel topic to get input linear and angular velocities from drive node.
- **turtlebot3_lds**: Existing node upon turtlebot bring up. This node publishes to /scan topic for lidar node to subscribe to to read in lidar scan data.
- **RVIZ**: this not technically not a node but it reads /visualization_marker topic published by pose node to plot trajectory of robot.

UML Class diagrams

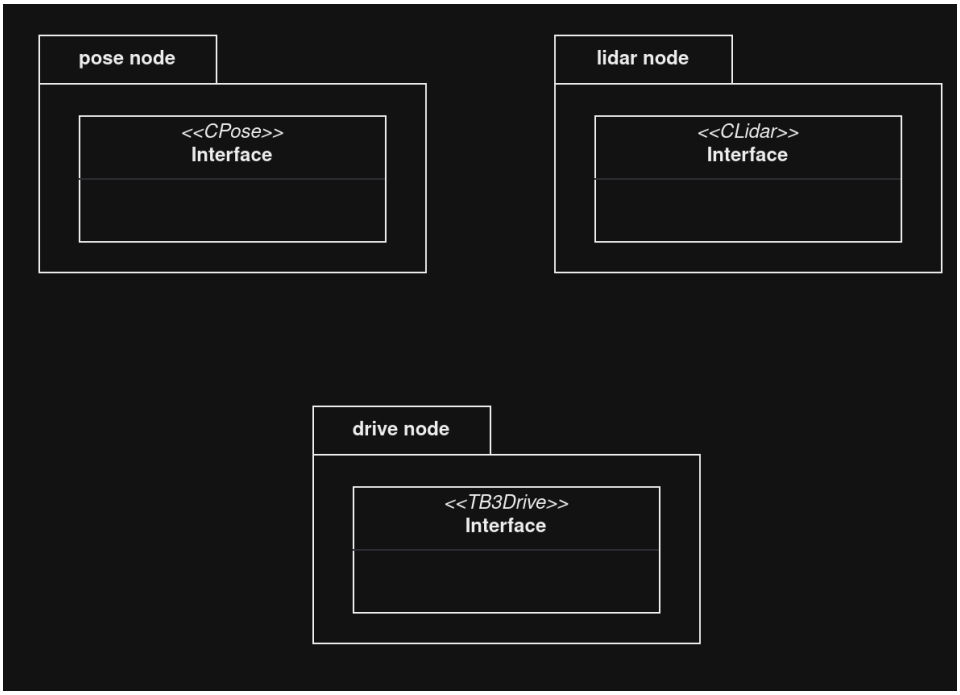
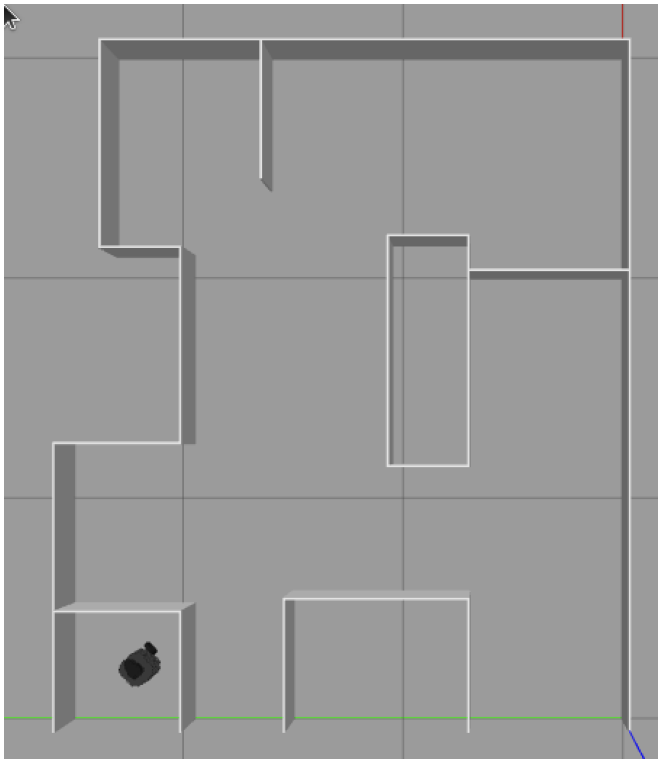


Figure 2: UML Class diagram

- **pose node**: subscribes to /odom topic and computes position of turtlebot and publishes to drive node. It also publishes visualization_marker message for RVIZ to plot trajectory.
- **lidar node**: subscribe to /scan topic to get lidar scan data and publishes to its own topic for drive node to subsrcibe to.
- **drive node**: publishes linear and angular velocities based on control algorithm to drive the turtlebot. It listens to pose and lidar nodes' topics to perform its control task.

Functionality and Testing



(a) simulated maze

(b) Trajectory plot of simulation

Figure 4: Picture of real maze

Real world maze matches simulated maze's layout.

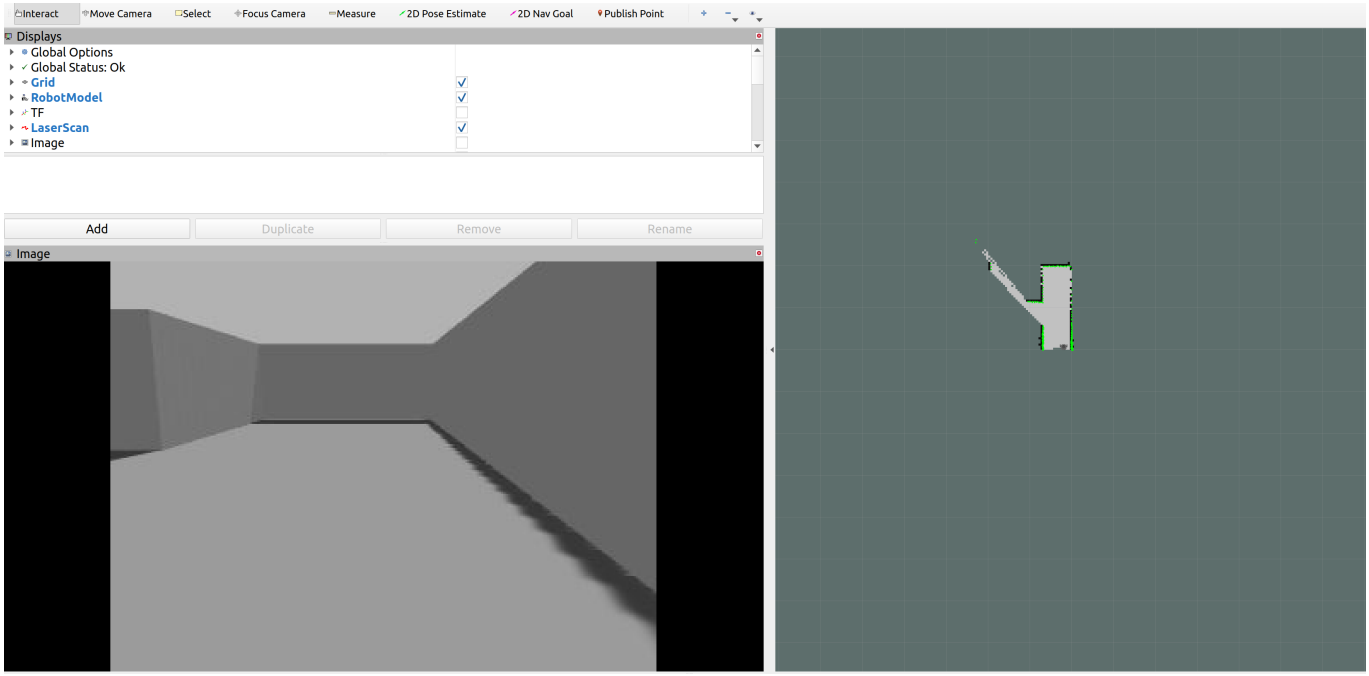


Figure 5: Camera module successful activation

Revision Control

```
2ef7377 (HEAD -> master, origin/master, origin/HEAD) FINAL RELEASE YIPEEEEE
66ad5b9 ADDed visualisation marker function to plot trajectory in simulation
fda9c6d FINAL OOP CODE RELEASE
a755639 final code, need run at simulation
f13d9b0 OOP Project push in for simulation and rvis
5b9e00d File messed up. Reorganise to include both NON oop and OOP design.
3084ca1 Update README.md
3526106 Adding all mazes to new code
a552c49 Final maze
bb0c4f1 push in pdf printout of refactored codes
feaafbb Added missing comments for Drive class header files
b2e8da2 Reorganise folders to contain both NON oop and OOP refactored code. Push
   pre final release for both versions.
bbd658f code works, porting to refactor code
eee4784 update on algorithm
815ca94 Maze 1 and 3
7455584 Merge branch 'master' of https://github.com/TomNgn3108/MTRX3760-Project01
d342902 Mazes
91e1352 Refactor code base push in, control loop algorithm in PROGRESS. Port over
    later
005dbf5 Package push
9eec591 basic instructions
be23fd1 Wall simulation code
ac5e6e7 initial
```

Teamwork

460311650

- Camera module
- Main control loop algorithm refine
- Real world maze testing
- hardware setup (turtelbot bring up debugging)
- Report writing

460418627

- Control algorithm
- Real world maze testing
- Report writing
- Real maze building

500554025

- ROS Node Design
- UML Diagram
- Creation of maze simulations
- Simulation testing with control algorithms
- Refining of algorithm design
- Real World Testing
- Report writing

500562776

- Code Refactoring into separate packages and nodes while preserving functionality.
- Simulation testing with prototype control algorithms
- Real World Testing
- Report writing
- Handled Git revision control
- RVIZ- trajectory plotting

Self-Assessment

Component	Estimated Grade
Functionality and Testing	27/30
Code Quality and Development Process	25/30
Design	32/40
Total	84/100

Code Appendix

src/sensor/include/CLidar.h

```
// This header file is for class CLidar which has the capability to
// read lidar data and publish it to TB3Drive class.
// The purpose of this is to isolate capability of reading in Lidar
// values from LaserScan msg so that any error in reading in lidar
// can be identified easily -- Follow OOP design

#ifndef CLIDAR_H_
#define CLIDAR_H_

#include <ros/ros.h>

#include <sensor_msgs/LaserScan.h>
#include <geometry_msgs/Twist.h>
#include <nav_msgs/Odometry.h>
#include <vector>
#include <std_msgs/Float64MultiArray.h>

const char TOPIC_NAME[] = "LIDAR";
const int LIDAR_DATA_SIZE = 3;
const int SCAN_ANGLE[] = {0, 45,315};

// Set queue size big to prevent loss if any
// delay occurs
//https://stackoverflow.com/questions/56444248/reason-to-set-queue-size-of-ros-publisher-or-subscriver-to-a-large-value
const int QSize = 1000;

/// @brief---
// CLidar interface-----------------------------------------------------
// This class is for storing the lidar data and publish to its own topic
// for drive class to listen to and get the data. It serves as a class
// that subscribes to the internal lidar scan topic and store it in a
// vector.
class CLidar{
  public:
    CLidar();
    ~CLidar();
    void LidarScanMsgCallBack(const sensor_msgs::LaserScan::ConstPtr &msg);
    void FillPublishData();

    private:
        // ROS NodeHandle
    ros::NodeHandle nh_;
    ros::NodeHandle nh_priv_;

    // ROS Subscriber to listen in lidar
    ros::Subscriber laserScanSub;

    // ROS publisher to publish to a new topic
    ros::Publisher lidarPub;

        // Data for publishing
    std::vector<double> ScanData;
    std_msgs::Float64MultiArray msg2Pub;
};

```

```
56 #endif
```

src/sensor/src/CLidar.cpp

```cpp
#include "CLidar.h"

// Implementation file for class CLidar
// Functions :
//            - Constructor
//            - Destructor
//            - Call back function sub to LaserScan msg
//            - Publshing function

//---Constructor
CLidar::CLidar():nh_priv_("~")
{
  ROS_INFO("Lidar Node initalised");
    // Initialise subscriber
  laserScanSub = nh_.subscribe("scan", QSize, &CLidar::LidarScanMsgCallBack, this);

  //ROS publisher to publish to a new topic
  lidarPub= nh_.advertise<std_msgs::Float64MultiArray>(TOPIC_NAME,QSize);

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

  // Populate publishing message Float64MultiArray
  // https://answers.ros.org/question/226726/push-vector-into-multiarray-message-and-publish-it/
  // set up dimensions
  msg2Pub.layout.dim.push_back(std_msgs::MultiArrayDimension());
  msg2Pub.layout.dim[0].size = ScanData.size();
  msg2Pub.layout.dim[0].stride = 1;
  msg2Pub.layout.dim[0].label = "x"; // or whatever name you typically use to index vec1
  msg2Pub.data.clear();
  ROS_ASSERT(true);
}

//---Destructor
CLidar::~CLidar()
{
  ScanData.clear();
  ScanData.empty();
  ros::shutdown();
}

//---Call back function sub to LaserScan msg
void CLidar::LidarScanMsgCallBack(const sensor_msgs::LaserScan::ConstPtr &msg)
{
  // Read in range of lidar measurement at specified angles
  for (int num = 0; num < LIDAR_DATA_SIZE ; num++)
  {
    if (std::isinf(msg->ranges.at(SCAN_ANGLE[num])))
    {
      ScanData[num] = msg->range_max;
```

```
    }
    else
    {
      ScanData[num] = msg->ranges.at(SCAN_ANGLE[num]);
    }

    // Infinite range
    if( ScanData[num]==0.0)
    {
      ScanData[num] = msg->range_max;
    }
  }
}

//---Publshing function
void CLidar::FillPublishData()
{
  // copy in the data
  for (int i = 0; i < LIDAR_DATA_SIZE; i ++){
    msg2Pub.data.push_back(ScanData[i]);
  }

  // Publish
  lidarPub.publish(msg2Pub);

  // Clear data
  ScanData.clear();
  msg2Pub.data.clear();

}

//---------------------------------------------------------------
// CLidar NODE
int main(int argc, char* argv[])
{
  ros::init(argc, argv, "Lidar");
  CLidar Lidar;
  ros::Rate loop_rate(500);

  while(ros::ok)
  {
    Lidar.FillPublishData();

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

src/bot/include/CPose.h

```
1  // This header file is for class CPosewhich has the capability to
2  // calculate position from odom msgs
3  // It publishes pose to Drive class . The purpose of this is to isolate
4  // capability of reading in odom values and calculate position value so
5  // that errors can be identified easily -- Follow OOP design
6  // Added functionality: this class can be also
7  #ifndef CPOSE_H_
8  #define CPOSE_H_
9  #include <ros/ros.h>
10 #include <sensor_msgs/LaserScan.h>
11 #include <geometry_msgs/Twist.h>
12 #include <geometry_msgs/Pose.h>
13 #include <nav_msgs/Odometry.h>
14 #include <nav_msgs/Path.h>
15 #include <visualization_msgs/Marker.h>
16
17
18 #include <vector>
19 #include<std_msgs/Float64.h>
20
21 const char trajectoryTopic[] = "visualization_marker";
22 const char topicName[] = "POSE";
23 // Set queue size big to prevent loss if any
24 // delay occurs
25 //https://stackoverflow.com/questions/56444248/reason-to-set-queue-size-of-ros-
   publisher-or-subscriver-to-a-large-value
26 const int QSize = 1000;
27
28 /// @brief---
29 // CPose interface-----------------------------------------------------
30 // This class subscribes to Odometry and acquire pose of the bot. It then
31 // publish turtlebot pose to Drive class for motion planning. It also stores
32 // current linear and angular velocity for other uses (not yet identified,
33 // but kept for debugging).
34 class CPose{
35   public:
36     CPose();
37     ~CPose();
38     void odomMsgCallBack(const nav_msgs::Odometry::ConstPtr &msg);
39     void PublishPose();
40     void TrajectoryVisualise();
41
42   private:
43     // ROS NodeHandle
44     ros::NodeHandle nh_;
45     ros::NodeHandle nh_priv_;
46
47     // Subscriber to odometry
48     ros::Subscriber odomSub;
49
50     // Publisher
51     ros::Publisher botPub;
52     ros::Publisher TrajectoryPub;
53
54     // Pose message
55     double tb3Pose;
```

```
56
57      // Store Pose
58      geometry_msgs::Pose odomPose;
59      std_msgs::Float64 msg;
60
61      //trajectory plot msgs
62      visualization_msgs::Marker trajectoryMsg;
63
64  };
65
66  #endif
```

src/bot/src/CPose.cpp

```
#include "CPose.h"

// Implementation file for class CLidar
// Functions :
//            - Constructor
//            - Destructor
//            - Call back function sub to odom msg
//            - Pose Publshing function
//            - Trajectory plotting function

//---Constructor
CPose::CPose():nh_priv_("~")
{
  ROS_INFO("Pose node initalised");
  // Subscribe to odometry topic
  odomSub = nh_.subscribe("odom", QSize, &CPose::odomMsgCallBack, this);

  //ROS publisher to publish to a new topic
  botPub = nh_.advertise<std_msgs::Float64>(topicName,QSize);
  TrajectoryPub = nh_.advertise<visualization_msgs::Marker>(trajectoryTopic,
QSize);

  // Pose data from odometry
  tb3Pose= 0.0;
  ROS_ASSERT(true);
}

//--- Destructor
CPose::~CPose()
{
  trajectoryMsg.points.clear();
  ros::shutdown;
}

//--- Call back function sub to odom msg
void CPose::odomMsgCallBack(const nav_msgs::Odometry::ConstPtr &msg)
{
  // Compute current odometry
  double siny = 2.0 * (msg->pose.pose.orientation.w * msg->
pose.pose.orientation.z + msg->pose.pose.orientation.x * msg->
pose.pose.orientation.y);
    double cosy = 1.0 - 2.0 * (msg->pose.pose.orientation.y * msg->
pose.pose.orientation.y + msg->pose.pose.orientation.z * msg->
pose.pose.orientation.z);

    tb3Pose = atan2(siny, cosy);
  odomPose = msg->pose.pose;
}

//---Publshing function
void CPose::PublishPose()
{
  msg.data = tb3Pose;
  botPub.publish(msg);
}

//--- Function publishes visualisation markers for path plot
```

```
53 void CPose::TrajectoryVisualise()
54 {
55   // TRAJECTORY MSGS
56   //http://wiki.ros.org/rviz/Tutorials/Markers%3A%20Basic%20Shapes
57   // Modify trajectory msg for publhsing
58   trajectoryMsg.header.frame_id = "map";            // which frame to use
59   trajectoryMsg.header.stamp = ros::Time();         // timing
60   trajectoryMsg.frame_locked = true;                // Lock frame
61
62   // set id and namespace for rViz to access points to plot
63   trajectoryMsg.ns = "points";
64   trajectoryMsg.id = 0;
65
66   // Set shape type of points and  tell msgs to add points
67   // on rviz
68   trajectoryMsg.type = visualization_msgs::Marker::POINTS;
69   trajectoryMsg.action = visualization_msgs::Marker::ADD;
70
71   // Set initial position of trajectory on the map
72   trajectoryMsg.pose.position.x = 0.0;
73   trajectoryMsg.pose.position.y = 0.0;
74   trajectoryMsg.pose.position.z = 0.0;
75
76   // Set initialorienation of trajectory on the map
77   trajectoryMsg.pose.orientation.x = 0.0;
78   trajectoryMsg.pose.orientation.y = 0.0;
79   trajectoryMsg.pose.orientation.z = 0.0;
80   trajectoryMsg.pose.orientation.w = 1.0;
81
82   // Size of plot points
83   trajectoryMsg.scale.x = 0.05;
84   trajectoryMsg.scale.y = 0.05;
85   trajectoryMsg.scale.z = 0.05;
86
87   // Set color of plot  - BLUE
88   trajectoryMsg.color.r = 0.0f;
89   trajectoryMsg.color.g = 0.0f;
90   trajectoryMsg.color.b = 1.0f;
91   trajectoryMsg.color.a = 1.0;
92
93   // Mesh resoruce
94   trajectoryMsg.mesh_resource = "package://pr2_description/meshes/base_v0
   /base.dae";
95
96   // Let points plotted exist on the map as long as simulation is running
97   trajectoryMsg.lifetime = ros::Duration();
98
99   // store list of odom pose.position
100  trajectoryMsg.points.push_back(odomPose.position);
101  TrajectoryPub.publish(trajectoryMsg);
102 }
103 //------------------------------------------------------------
104 // CPose NODE
105 int main(int argc, char* argv[])
106 {
107   ros::init(argc, argv, "Pose_Node");
108   CPose bot;
109   ros::Rate loop_rate(500);
110
```

```
111   while(ros::ok)
112   {
113     bot.PublishPose();
114     bot.TrajectoryVisualise();
115     // process callback for this node
116     ros::spinOnce();
117     loop_rate.sleep();
118   }
119
120   return 0;
121 }
```

src/drive/include/TB3Drive.h

```
1  // Original class was turtlebot_drive taken from simulation example package
2  // Class TB3Drive is the main class that controls the bot based on
3  // lidar and odom data published by CLidar and CPose nodes.
4
5  #ifndef TB3DRIVE_H_
6  #define TB3DRIVE_H_
7
8  #include <ros/ros.h>
9
10 #include <sensor_msgs/LaserScan.h>
11 #include <geometry_msgs/Twist.h>
12 #include <nav_msgs/Odometry.h>
13 #include <vector>
14 #include <std_msgs/Float64MultiArray.h>
15 #include <std_msgs/Float64.h>
16
17
18 // Lidar indexing
19 const int CENTER = 0;
20 const int LEFT   = 1;
21 const int RIGHT  =  2;
22
23 // Velocity
24 const double STOP_FOWARD_V = 0.0;
25
26 // Bot states
27 const int STRAIGHT= 0;
28 const int LEFT_TURN = 1;
29 const int CORNER_TURN   = 2;
30 const int DEFAULT_STATE = 3;
31
32
33 // TB3DRive interface-------------------------------------------------------
34 // This class controls the robot based on the lidar readings. The class has the
35 // capabilities to transit states of the robot and compute linear and angular
36 // velocities and publish to cmd_vel to set velocity of the bot.
37
38 class TB3Drive
39 {
40  public:
41   TB3Drive();
42   ~TB3Drive();
43   bool controlLoop();
44
45  private:
46   // ROS NodeHandle
47   ros::NodeHandle nh_;
48   ros::NodeHandle nh_priv_;
49
50   // ROS Topic Publishers
51   ros::Publisher cmd_vel_pub_;
52
53   // ROS Topic Subscribers
54   ros::Subscriber cLidarSub;
55   ros::Subscriber cBotSub;
56
```

```
57    double forwardTarget;
58    double forwardTargetTurn;
59    double sideTarget;
60
61    double maxTurnVel;
62
63    double maxForwardVel;
64    double minForwardVel;
65
66    double angularVel;
67    double linearVel;
68
69    double turnKp;               // Proportional gain for angular velocity
70    double forwardKp;            // Proportional gain for linear velocity
71
72    double tb3Pose;             // Current Position form odometry - sent to by
  CPose
73    double prevTB3pose;         // Previous Position from odometry
74
75    int leftTurnFlag;
76
77    std::vector<double>lidarData;
78
79    // Function publishes to cmd_vel topic to control linear
80    // and angular velocity of turtlebot.
81    void updatecommandVelocity(double linear, double angular);
82
83    // Callback functions receiving messages from CPose class and CLidar class
84    void cLidarMsgCallBack(const std_msgs::Float64MultiArray::ConstPtr &msg);
85    void cPoseMsgCallBack(const std_msgs::Float64::ConstPtr &msg);
86 };
87 #endif
88
```

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 }
```