Lab 4 – Robot Trajectory and Velocity

By now you have worked with ROS and got some basic knowledge. In this Lab, you will learn how to control robot movements, and how to record odometry and velocity data for creating trajectory and velocity graphs. Note that only odometry data is used for three tasks in this Lab.

4.1 Control the robot movement using odometry data

Task 4.1 You should use odometry data to control your robot to move forward, go through the 1st gap and the 2nd gap, and reach to the charger position, as shown in Figure 4.1.

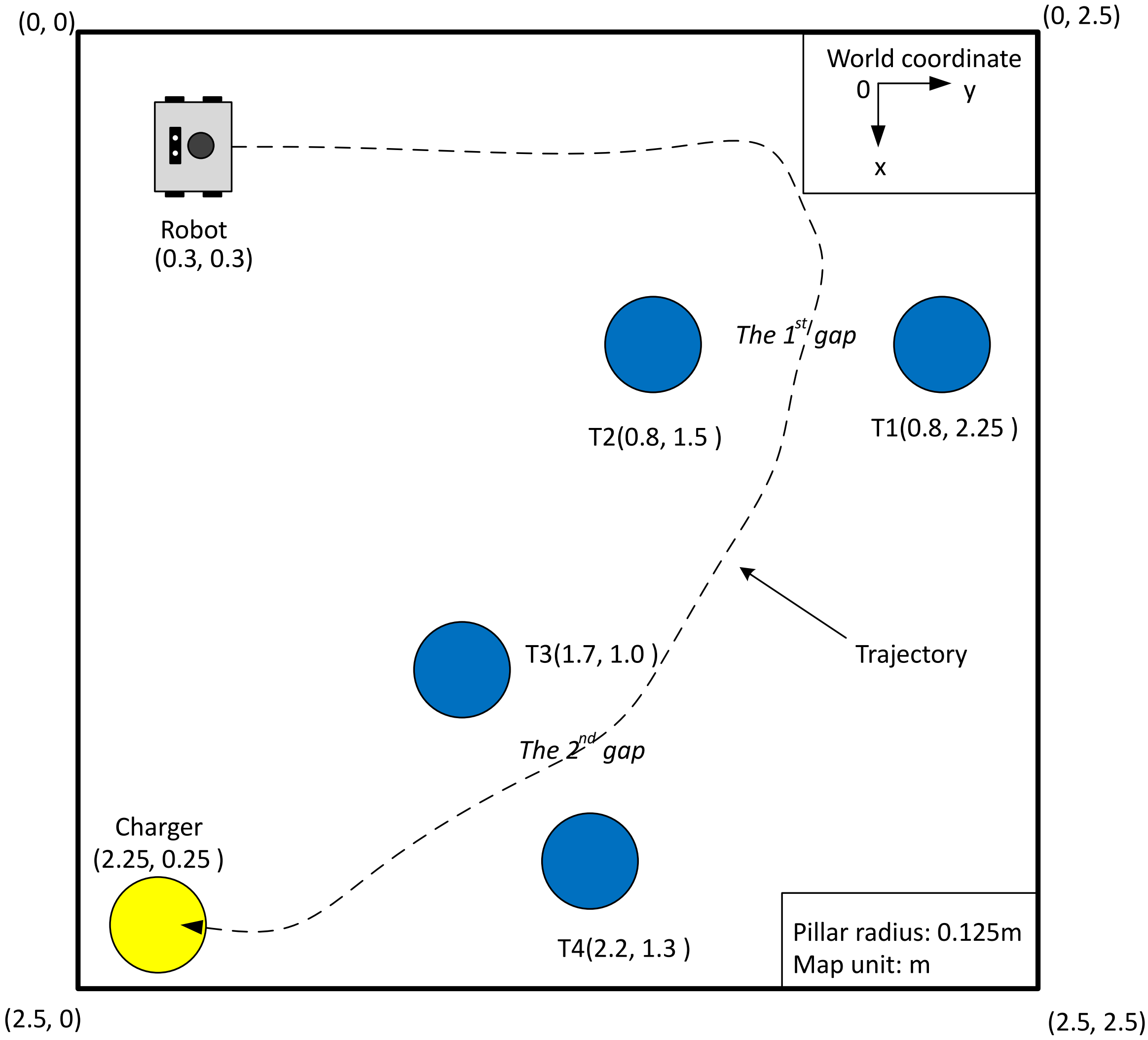


Figure 4.1 The map for the mobile robot to navigate

At the 1st stage, you could control your robot move forward along a straight line to the position (0.3, 1.1) and then gradually turn right to reach the position (0.8, 1.875), i.e. the middle between pillars T1 and T2.

Step 1: Go to the following folder.

cd ~/ros_workspace/src/tutorial_pkg/src

Step 2: Using *gedit* to open *tutorial_pkg_node.cpp*. Then remove some codes with Green colour and add some codes with Yellow colour.

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "nav_msgs/Odometry.h"

using namespace std;

class RobotMove {  // main class public:
    // Turnable parameters
 constexpr const static double FORWARD_SPEED_LOW = 0.1;
 constexpr const static double FORWARD_SPEED_MIDDLE = 0.3;
 constexpr const static double FORWARD_SPEED_HIGH = 0.5;
 constexpr const static double FORWARD_SPEED_STOP = 0;
 constexpr const static double TURN_LEFT_SPEED_HIGH = 1.0;
constexpr const static double TURN_LEFT_SPEED_MIDDLE = 0.6;
 constexpr const static double TURN_LEFT_SPEED_LOW = 0.3;
 constexpr const static double TURN_RIGHT_SPEED_HIGH = -1.0;
 constexpr const static double TURN_RIGHT_SPEED_MIDDLE = -0.6;
 constexpr const static double TURN_RIGHT_SPEED_LOW = -0.3;
 RobotMove();     void startMoving();    void moveStop();      void
moveForward(double forwardSpeed);    void moveRight(double
turn_right_speed);
    void moveForwardRight(double forwardSpeed, double turn_right_speed);
    // add the following code void odomCallback(const
    nav_msgs::Odometry::ConstPtr& odomMsg);
private:
 ros::NodeHandle node;    ros::Publisher commandPub; // Publisher to the robot's
    ros::Subscriber odomSub;     //Subscriber to robot's odometry topic
    double PositionX=0.3, PositionY=0.3, landmark1=1.15, landmark2=0.9;
velocity command topic
    double homeX = 0.3, homeY = 0.3;
};

RobotMove::RobotMove(){
    //Advertise a new publisher for the simulated robot's velocity command topic at 10Hz
    commandPub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);
   // subscribe to the odom topic            odomSub = node.subscribe("odom",
20, &RobotMove::odomCallback, this);
}

//send a velocity command
void RobotMove::moveForward(double forwardSpeed){
 geometry_msgs::Twist msg;//The default constructor to set all commands to 0
 msg.linear.x = forwardSpeed; //Drive forward at a given speed along the x-axis.
    commandPub.publish(msg);
}
```

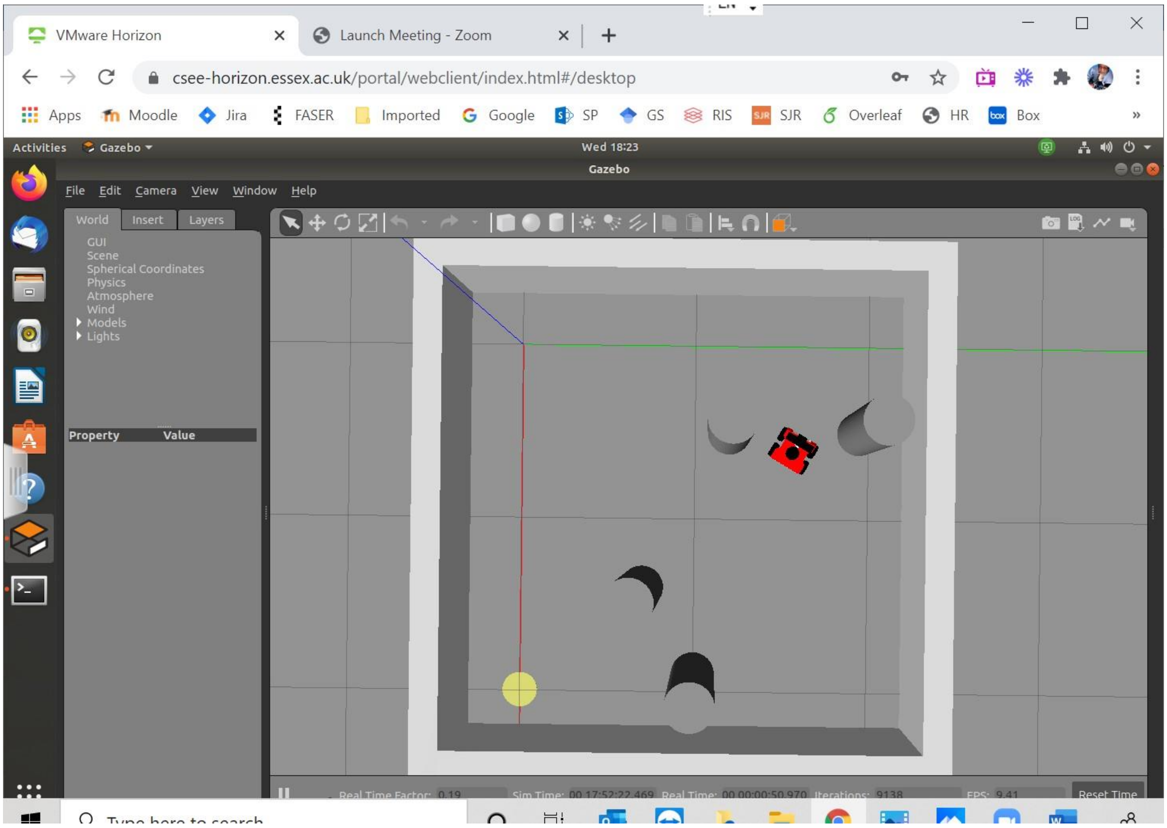
```
void RobotMove::moveStop(){
 geometry_msgs::Twist msg;     msg.linear.x =
FORWARD_SPEED_STOP;
      commandPub.publish(msg);
}

void RobotMove::moveRight(double turn_right_speed){
 geometry_msgs::Twist msg;     msg.angular.z
= turn_right_speed;
      commandPub.publish(msg);
}

// remove the following code void RobotMove::startMoving(){
 ros::Rate rate(20);    //Define rate for repeatable operations.
      ROS_INFO("Start moving");
// keep spinning loop until user presses Ctrl+C
 while (ros::ok()){
     // Check if ROS is working. If ROS master is stopped or there was sent signal //
    to stop the system, ros::ok() will return false.
                  moveForward(FORWARD_SPEED_LOW);
        ROS_INFO_STREAM("Robot speed: " << FORWARD_SPEED_LOW);
 ros::spinOnce(); // Allow ROS to process incoming messages               rate.sleep();
// Wait until defined time passes.
       }
}

// add the following code
void RobotMove::moveForwardRight(double forwardSpeed, double turn_right_speed){
            //move forward and right at the same time
            geometry_msgs::Twist msg; msg.linear.x
            = forwardSpeed; msg.angular.z =
            turn_right_speed;
            commandPub.publish(msg);
}

// add the callback function to determine the robot position.
void RobotMove::odomCallback(const nav_msgs::Odometry::ConstPtr& odomMsg){
            PositionX = odomMsg->pose.pose.position.x + homeX;
            PositionY = odomMsg->pose.pose.position.y + homeY;
}

// add the following function void RobotMove::startMoving(){
 int stage = 1;       ros::Rate rate(20);    //Define rate for
repeatable operations.       ROS_INFO("Start moving");
```

```
 while (ros::ok()){    // keep spinning loop until user presses Ctrl+C
switch(stage){
    case 1: // the robot move forward from home
if (PositionY < landmark1)
                moveForward(FORWARD_SPEED_HIGH);
else stage = 2;
              break;
    case 2: // the robot turns right toward the 1st gap             if (PositionX < landmark2)
moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
               else stage = 3;
               break;
     case 3: // the robot stops at the middle of the 1st gap
moveStop();
               break;
     }          ros::spinOnce(); // Allow ROS to process incoming
messages          rate.sleep();   // Wait until defined time passes.
    }
}

int main(int argc, char **argv) {
      ros::init(argc, argv, "RobotMove");   // Initiate new ROS node named "stopper"
      RobotMove RobotMove;                  // Create new RobotMove object
 RobotMove.startMoving();           // Start the movement       return
0;
}
```

Step 3: To compile and run your code.

```
cd  ~/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

You will see that the robot firstly travels along a straight line path, then turns toward the middle of Gap 1 and finally stops in the middle between the pillars T1 and T2. Figure 3.2 shows the final result.



Step 5: To enable the robot to continue its movement until reaching the charger, you should add the following code in Yellow colour.

```
class RobotMove { // main class public:
    // some existing code private:
    // some existing code
    double landmark3 = 1.2, landmark4 = 1.83, landmark5 = 2.25;
}    void RobotMove::startMoving(){ ros::Rate rate(20);
//Define rate for repeatable operations.    ROS_INFO("Start
moving");
  while (ros::ok()){ // keep spinning loop until user presses Ctrl+C
  switch(stage){
    case 1: // the robot move forward from home
  if (PositionY < landmark1)
          moveForward(FORWARD_SPEED_HIGH);
  else stage = 2;
          break;
    case 2: // the robot turns right toward the 1st gap
  if (PositionX < landmark2)
       moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
  else stage = 3;
          break;
       case 3: // the robot moves forward fast
          if (PositionX < landmark3)
```

```
                moveForward(FORWARD_SPEED_HIGH);
               else stage = 4;
    break;
            case 4: // the robot moves and turns right slowly
          if (PositionX < landmark4)
              moveForwardRight(FORWARD_SPEED_MIDDLE,TURN_RIGHT_SPEED_MIDDLE);
    else stage = 5;          break;
            case 5: // the robot moves towards the charger
        if (PositionX < landmark5)
                moveForward(FORWARD_SPEED_HIGH);
            else stage = 6;
              break;
        case 6: // stop at the charger position
    moveStop();
              break;
        }          ros::spinOnce(); // Allow ROS to process incoming
    messages          rate.sleep();   // Wait until defined time passes.
       }
    }
```

Step 6: You can compile the new code and run it using following commands.

```
cd  ~/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

You will see the robot is able to reach to the charger position. Also, you may change these stages to see how the movement of the robot is changing.

4.2 Odometry data for creating trajectory graph

To obtain the odometry message, you should include the following head file in your code:

```
#include "nav_msgs/Odometry.h"
```

As the position of the robot is published to the topic '/*odom*', you can subscribe to this topic to obtain the real-time position. You may use following command:

rostopic echo /odom to examine the topic content shown in Figure 3.3. Note that the position of the robot is:

```
PositionX = odomMsg->pose.pose.position.x;
PositionY    =    odomMsg->pose.pose.position.y;
```

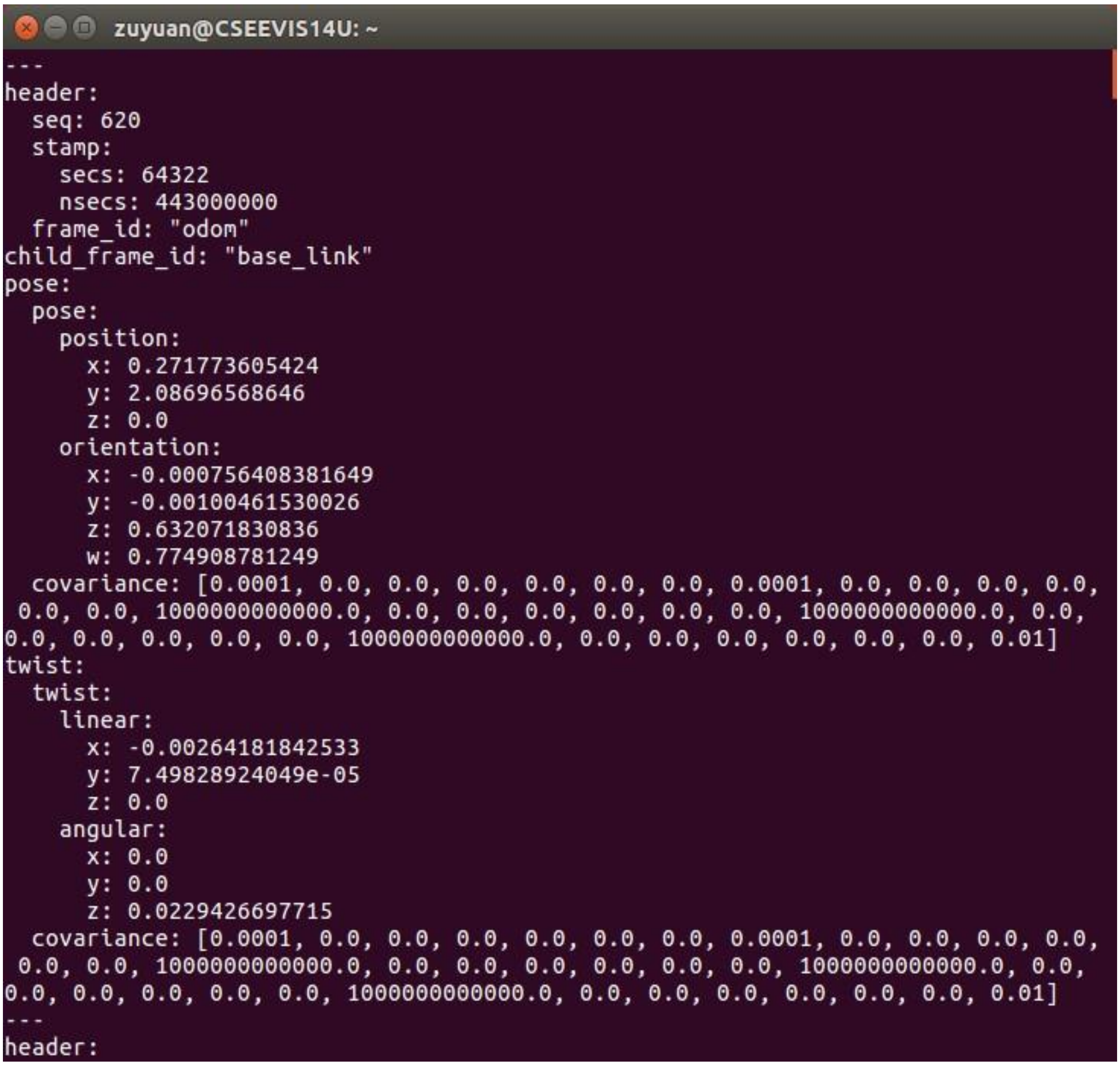
where *odomMsg* is the odometry message name defined by yourself.

To read the pose information, you need subscribe to the '/odom' topic:

odomSub = node.subscribe("odom", 20, &RobotMove::odomCallback, this) Also, you should define the *odomCallback* function in order to get whatever you want.

For instance, in order to obtain the robot trajectory you may define:

```
void RobotMove::odomCallback(const nav_msgs::Odometry::ConstPtr& odomMsg){
        PositionX = odomMsg->pose.pose.position.x;
        PositionY = odomMsg->pose.pose.position.y;
}
```



```
zuyuan@CSEEVIS14U: ~
---
header:
  seq: 620
  stamp:
    secs: 64322
    nsecs: 443000000
  frame_id: "odom"
child_frame_id: "base_link"
pose:
  pose:
    position:
      x: 0.271773605424
      y: 2.08696568646
      z: 0.0
    orientation:
      x: -0.000756408381649
      y: -0.00100461530026
      z: 0.632071830836
      w: 0.774908781249
  covariance: [0.0001, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0001, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.01]
twist:
  twist:
    linear:
      x: -0.00264181842533
      y: 7.49828924049e-05
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0229426697715
  covariance: [0.0001, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0001, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1000000000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.01]
---
header:
```

Figure 4.2 Message published from the '/odom' topic.

Task 4.2 Run your code and record your robot's trajectory in a csv file. Then you should use LibreOffice in Ubuntu or Excel in Windows to plot this trajectory starting from the home position and ending at the charger position.

The following code is required to complete this task.

Step 1: You should include the following code at the beginning of your code.

```
// some existing code #include
<fstream>
#include <time.h>
#include <iomanip>
```

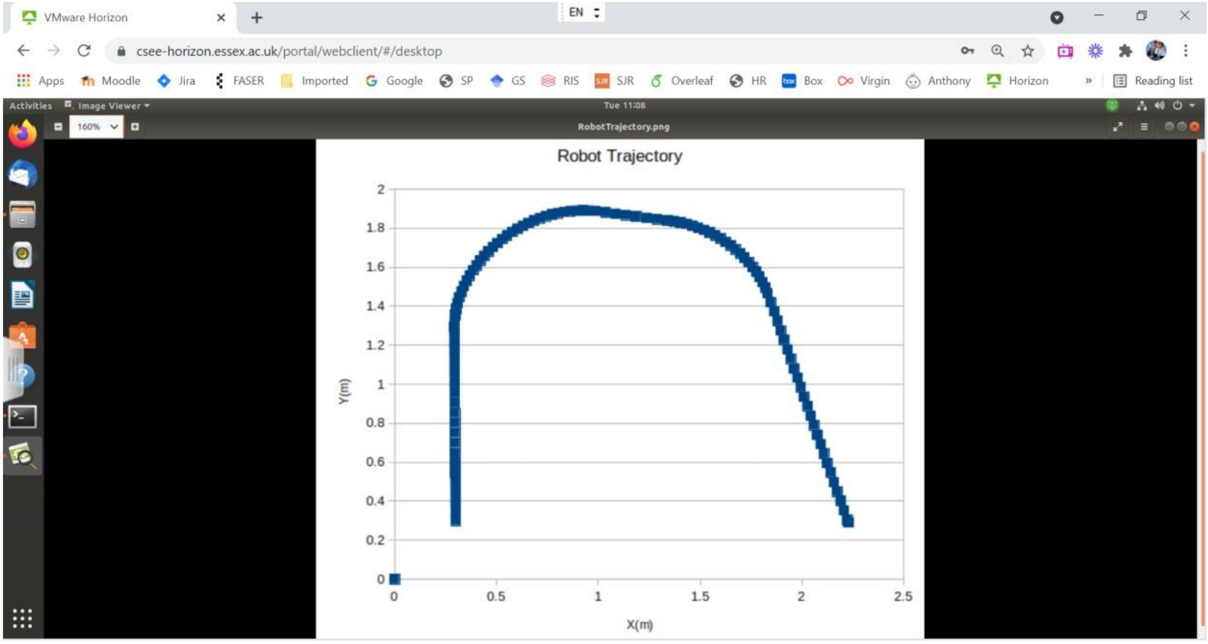
Step 2: To add the following Yellow code in your existing code.

```
ofstream openFile(const string& name){ // open files for data storage string
    homedir = getenv(HOME"); ostringstream path; path << homedir << "/M-
    Drive/ros_workspace/src/tutorial_pkg/" << name; return
    ofstream(path.str());
};

void RobotMove::startMoving(){
ofstream   odomTrajFile   =   openFile("odomTrajData.csv");
ros::Rate rate(20);       ROS_INFO(Start moving");
    while (ros::ok()){
    switch(stage){
case 1:

            // other existing code
                case 6:   // stop at the charger
        position
            moveStop();
            break;
        }
        // some existing code
    odomTrajFile << PositionX << " " << PositionY << endl;
ros::spinOnce();
        rate.sleep();
    }
    odomTrajFile.close();
}
```

Then, you can compile the new code and run it using following commands.

```
cd ~/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

After running your code, you will be able to plot the data in odomTrajData.csv using LibreOffice in Ubuntu or Excel in Windows. It should looks like the picture below.



4.3 Velocity graph

The velocity of the robot is also published to the *'/odom'* topic, e.g. *odomMsg->twist.twist.linear.x* is the velocity along the forward direction. Note: *odomMsg* is the odometry message name defined by yourself.

Task 4.3 Run your code and record your robot's velocity in a csv file. Then you should use Excel to plot a Speed-Time graph. The robot should start from the home position and stop at the charger position. Explain why the velocity graph is in such condition.

Step 1: You should add the following YELLOW code into your code

```
// some existing code….
#include <iostream>
#include <cmath>
#include <chrono>

using namespace std::chrono;

class RobotMove { public:
 // some existing code…… private:
 // some existing code……
 ros::Time current_time;
     ros::Duration real_time;
}
```

Step 2: To add the following code into class RobotMove{} *void RobotMove::startMoving(){*

```
    ofstream odomVelFile = openFile("odomVelData.csv");
auto currentTime = high_resolution_clock::now(); auto
startTime = currentTime;  // some existing code
 while(ros::ok()){         switch(stage){          case 1:
                 // some existing code
           case 6:        // stop at the charger position
              moveStop();
              break;
          }
          auto currentTime = high_resolution_clock::now();
duration<double,std::deca>runTime = currentTime – startTime;
          runtime *= 10; // convert time to seconds
      odomVelFile << ceil(runtime.count()) << " " << robVelocity << endl;
// some existing code
     }
     // some existing code odomVelFile.close();
}
```

Step 3: you can compile the new code and run it using following commands.

```
cd ~/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

Step 4: After running your code, you will be able to plot the data in odomVelData.csv using LibreOffice in Ubuntu or Excel in Windows. Explain the robot velocity you obtained.