Министерство науки и высшего образования РФ Федеральное государственное бюджетное образовательное учреждение высшего образования «Удмуртский государственный университет»

Институт гражданской защиты

Кафедра цифровых инженерных технологий

# Лабораторная работа №4

по дисциплине

Прикладное программное обеспечение

Выполнил студент

группы ОМ-20.04.01.04-11

Вологжанин Е. А.

Проверил к.т.н. доцент

Клековкин А. В

Оглавление

[Лабораторная работа №4 1](#_Toc163127127)

[Цель 3](#_Toc163127128)

[Задачи 3](#_Toc163127129)

[Теоретическая часть 3](#_Toc163127130)

[Практическая часть 6](#_Toc163127131)

[Задача 1 6](#_Toc163127132)

[Задача 2 7](#_Toc163127133)

[Вывод 8](#_Toc163127134)

[Приложение 1 8](#_Toc163127135)

# Цель

Реализовать движение робота по траектории, заданной функцией в параметрическом виде.

# Задачи

1. Реализовать движение робота по траектории с обратной связью;
2. Вывести графики теоретической и реальной траектории для каждого эксперимента. Вывести теоретические и реальные значения линейной и угловой скорости.

# Теоретическая часть

При решении поставленных задач стоит учитывать кинематическое соотношение – связь неподвижной и подвижной (привязана к корпусу робота) систем координат. Что бы полностью описать положение робота в пространстве необходимо 3 координаты: (X, Y, α).

|  |  |
| --- | --- |
|  |  |

Где a – ускорение, – угловая скорость, – угол поворота.

Вторая производная имеет вид:

|  |
| --- |
|  |
|  |

Стоит рассмотреть уравнения второй производной:

|  |
| --- |
|  |

Если в получившийся пример вместо подставить известные уравнения траектории, то получим управляющее воздействие для движения по этой траектории.

|  |
| --- |
|  |

(Желаемая траектория)

Тогда управляющее воздействие для движения по желаемой траектории будет выглядеть так:

|  |
| --- |
|  |

Динамика ошибки, в этом случае, задаётся как:

|  |
| --- |
|  |

Где – пропорциональный коэффициент, – дифференциальный коэффициент.

Необходимо подобрать такие коэффициенты, чтобы корни уравнения принадлежали множеству вещественных чисел (т.к. при получении комплексных чисел регулятор придаёт роботу колебания при попытке вернуться к нужному курсу).

Тогда управление задаётся уравнением:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

В итоге получились два уравнения управления, для получения линейной () и угловой () скоростей.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Практическая часть

## Задача 1

При реализации данной задачи был использован вариант пид-регулятора. Траектория задана следующим параметрическим уравнением, при зависимой переменной t∈ [0; 2π]:

|  |
| --- |
|  |
|  |

В качестве зависимой переменной было взято время из ROS.

!!!{Придумать скрин, возможно через RViz}

Листинг программы прикреплён в Приложении 1.

## Задача 2

Для вывода траекторий и скоростей робота была использован «rqt» – фреймворк на основе Qt для разработки графического интерфейса для ROS. Данный фреймворк предоставляет удобный плагин «rqt\_multiplot» – плагин графического интерфейса для визуализации числовых значений на нескольких 2D графиках.

Получившиеся результаты движения робота по траектории:

!!!{Скрины с rqt}

# Вывод

В ходе выполнения работы были реализованы задачи движения робота по заданной траектории с обратной связью, посредством написания программы на языке C++, и вывода теоретических и реальных скоростей робота.

# Приложение 1

|  |
| --- |
| *#include* <ros/ros.h>  *#include* <std\_msgs/Float32.h>  *#include* <geometry\_msgs/Twist.h>  *#include* <nav\_msgs/Odometry.h>  *#include* <signal.h>  *#include* <tf/transform\_broadcaster.h>  *#include* <vector>  *#define* PI 3.1415926  ros::Publisher pub;  ros::Publisher pubTheorTrajectory;  ros::Publisher pubRealLinVel;  ros::Publisher pubRealAngVel;  geometry\_msgs::Twist messageStop;  geometry\_msgs::Twist messageMotion;  geometry\_msgs::Vector3 points;  double linVelReal = 0;  double angVelReal = 0;  double xPosReal = 0;  double yPosReal = 0;  double angleReal = 0;  float k\_d = 5;  float k\_p = 4;  float time\_coef = 0.05;  float correct\_coef = 0.5;  void mySigintHandler(int *sig*)  {      pub.publish(messageStop);      ros::Duration(1).sleep();      ROS\_INFO("I'm stoped sigint");      ros::shutdown();  }  void setupPublishers(ros::NodeHandle &*nh*)  {      pub = nh.advertise<geometry\_msgs::Twist>("/cmd\_vel", 10);      pubTheorTrajectory = nh.advertise<geometry\_msgs::Vector3>("theor\_trajectory", 10);      pubRealLinVel = nh.advertise<geometry\_msgs::Vector3>("real\_lin\_vel", 100);      pubRealAngVel = nh.advertise<geometry\_msgs::Vector3>("real\_ang\_vel", 100);  }  void getAngle(const nav\_msgs::Odometry::ConstPtr &*msg*) {      angleReal = atan2(msg->pose.pose.orientation.z, msg->pose.pose.orientation.w) \* 2;  }  void msgCallbackOdom(const nav\_msgs::Odometry::ConstPtr &*msg*)  {      linVelReal = msg->twist.twist.linear.x;      angVelReal = msg->twist.twist.angular.z;      xPosReal = msg->pose.pose.position.x;      yPosReal = msg->pose.pose.position.y;      getAngle(msg);  }  void calculatePointsT(int *length*, float \* *arr*) {      double step = 2.0 \* PI / length;      double value = 0.0;  *for*(int i = 0; i < length; i++) {          arr[i] = value;          value += step;          ROS\_INFO("t=%f", arr[i]);      }  }  float X(float *t*)  {      float Shift = -2;  *return* (cos(t \* time\_coef) + cos(2\*t \* time\_coef) + Shift) \* correct\_coef;  }  float Xdt(float *t*)  {  *return* (-time\_coef\*sin(t \* time\_coef) - 2\*time\_coef\*sin(2\*t \* time\_coef)) \* correct\_coef;  }  float Xdtdt(float *t*)  {  *return* (-time\_coef\*time\_coef\*cos(t \* time\_coef) - 4\*time\_coef\*time\_coef\*cos(2\*t \* time\_coef)) \* correct\_coef;  }  float Y(float *t*)  {  *return* 2\*sin(t \* time\_coef) \* correct\_coef;  }  float Ydt(float *t*)  {  *return* 2\*time\_coef\*cos(t \* time\_coef) \* correct\_coef;  }  float Ydtdt(float *t*)  {  *return* -2\*time\_coef\*time\_coef\*sin(t \* time\_coef) \* correct\_coef;  }  void calculatePointsOfTrajectory(int *length*, float \* *x*, float \* *y*, float \* *t*) {  *for*(int i = 0; i < length; i++)      {          x[i] = X(t[i]/time\_coef);          y[i] = Y(t[i]/time\_coef);          points.x = x[i];          points.y = y[i];          ROS\_INFO("X=%f, Y=%f", x[i], y[i]);          pubTheorTrajectory.publish(points);          ros::Duration(0.01).sleep();          ros::spinOnce();      }  }  void CalculateMotionVel(double &*omega*, double &*acceleration*, double *t*)  {      float X\_Part =  Xdtdt(t) +                      k\_d \* (Xdt(t) - linVelReal \* cos(angleReal)) +                      k\_p \* (X(t) - xPosReal);      float Y\_Part =  Ydtdt(t) +                      k\_d \* (Ydt(t) - linVelReal \* sin(angleReal)) +                      k\_p \* (Y(t) - yPosReal);      acceleration = cos(angleReal) \* X\_Part + sin(angleReal) \* Y\_Part;      omega = cos(angleReal) / linVelReal \* Y\_Part - sin(angleReal) / linVelReal \* X\_Part;  }  void publishRealLinAngVel(int *count*, double *lin*, double *ang*) {      geometry\_msgs::Vector3 messageLinVel;      geometry\_msgs::Vector3 messageAngVel;      messageLinVel.x = count;      messageLinVel.y = lin;      pubRealLinVel.publish(messageLinVel);      messageAngVel.x = count;      messageAngVel.y = ang;      pubRealAngVel.publish(messageAngVel);  }  int main(int *argc*, char \**argv*[])  {      int numberPoints = 500;      float t\_points[numberPoints];      float x\_points[numberPoints];      float y\_points[numberPoints];      ros::init(argc, argv, "motion\_with\_pid\_node", ros::init\_options::NoSigintHandler);      ros::NodeHandle nh;      ros::Subscriber subOdom = nh.subscribe("odom", 10, msgCallbackOdom);      ros::spinOnce();      signal(SIGINT, mySigintHandler);      setupPublishers(nh);      calculatePointsT(numberPoints, t\_points);      calculatePointsOfTrajectory(numberPoints, x\_points, y\_points, t\_points);      messageMotion.angular.z = PI / 2 / 3;      pub.publish(messageMotion);      ros::Duration(3).sleep();      ros::spinOnce();      messageMotion.angular.z = 0;      pub.publish(messageMotion);      double omega, acceleration, timeReal, timeRealOld, velocity, dtReal = 0.0;      double startTime = ros::Time::now().toSec();      int count = 0;  *while* (timeReal < 2.0 \* PI / time\_coef)      {          messageMotion.linear.x = velocity;          messageMotion.angular.z = omega;          pub.publish(messageMotion);          publishRealLinAngVel(count, velocity, omega);          ros::Duration(0.05).sleep();          ros::spinOnce();          CalculateMotionVel(omega, acceleration, timeReal);          count++;          timeReal = ros::Time::now().toSec() - startTime;          dtReal = timeReal - timeRealOld;          timeRealOld = timeReal;          velocity = velocity + acceleration \* dtReal;          ROS\_INFO("V=%f, W=%f", velocity, omega);      }      mySigintHandler(1);  *return* 0;  } |