Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Удмуртский государственный университет»

Институт гражданской защиты

Кафедра цифровых инженерных технологий

Лабораторная работа по дисциплине

Прикладное программное обеспечение

**Движение вдоль траектории**

Выполнил студент

группы ОМ-20.04.01.04-11

Вологжанин Егор А.

Оглавление

[Цель 3](#_Toc157179791)

[Задачи 3](#_Toc157179792)

[Часть 1. Теория 3](#_Toc157179793)

[Схема робота, Кинематические соотношения 3](#_Toc157179794)

[Движение робота по траектории 4](#_Toc157179795)

[Алгоритм 5](#_Toc157179796)

[Формулы для движения с обратной связью 5](#_Toc157179797)

[Часть 2. Результаты 8](#_Toc157179798)

[Функция для реализации 8](#_Toc157179799)

[Движение по траектории без обратной связи 8](#_Toc157179800)

[Движение по траектории с обратной связью 10](#_Toc157179801)

[Часть 3. Реализация кода программы 11](#_Toc157179802)

# Цель

Разработка алгоритма управления движением мобильного робота вдоль заданной траектории в среде ROS и Gazebo. Реализовать движение роботов по траектории, заданной функцией в параметрическом виде, используя ROS и Gazebo.

# Задачи

* реализовать программное движение робота по траектории.
* реализовать движение робота по траектории с обратной связью.
* вывести графики теоретической и реальной траектории для каждого эксперимента с помощью rqt-multiplot; (вывести теоретические и реальные значения линейной и угловой скорости).

# Часть 1. Теория

## \documentclass{article} \usepackage{amsmath} \pagestyle{empty} \begin{document} \begin{gather*} \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{\alpha} \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \omega \end{pmatrix}; \end{gather*} \end{document}Схема робота, Кинематические соотношения

— неподвижная система координат;

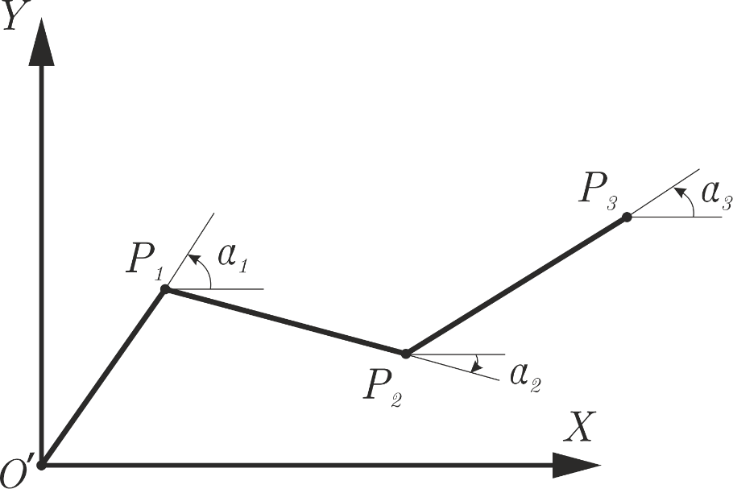
— подвижная система координат, жестко связанная с корпусом робота;

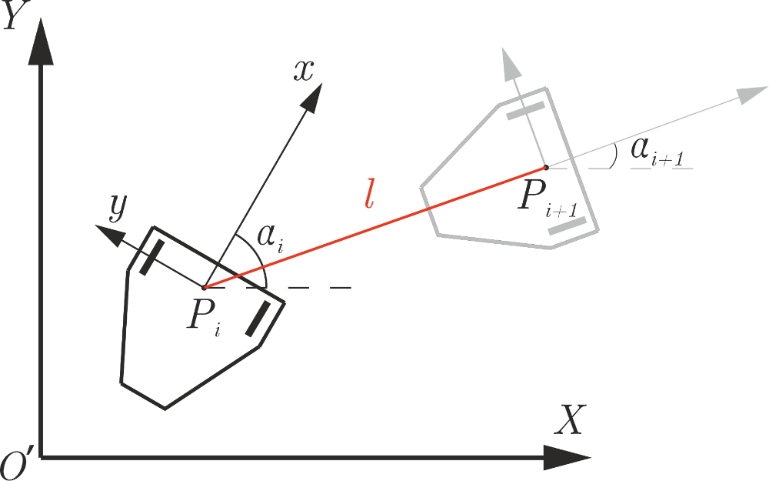
— координаты начала подвижной системы координат;

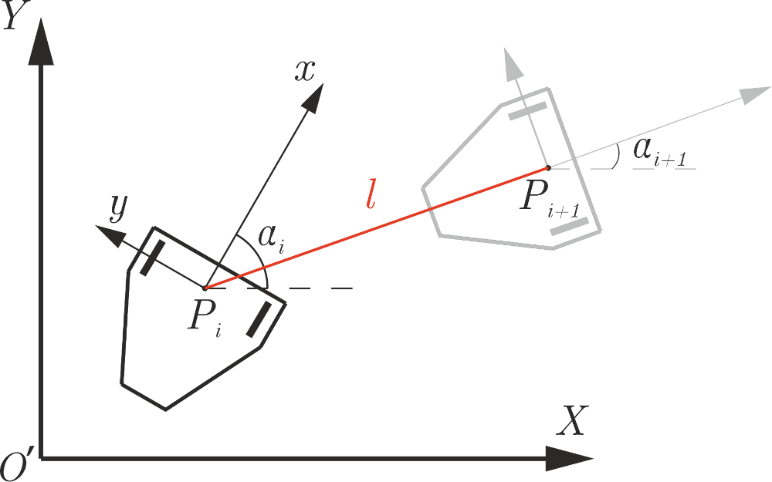
— угол поворота подвижной системы координат относительно неподвижной;

— векторы линейной и угловой скорости робота, спроецированные на подвижную систему координат.

## Движение робота по траектории

При движении робота по траектории, ориентация робота в каждой точке сонаправлена с касательной к траектории в этой точке.

Для траектории вида X(t) = f(t), Y(t) = g(t), можно найти точки Pi и углы αi. Pi = (Xi;Yi); (.

Расстояние до следующей точки ; линейная и угловая скорости , .

## Алгоритм

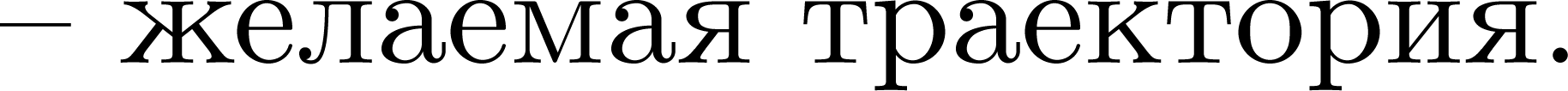
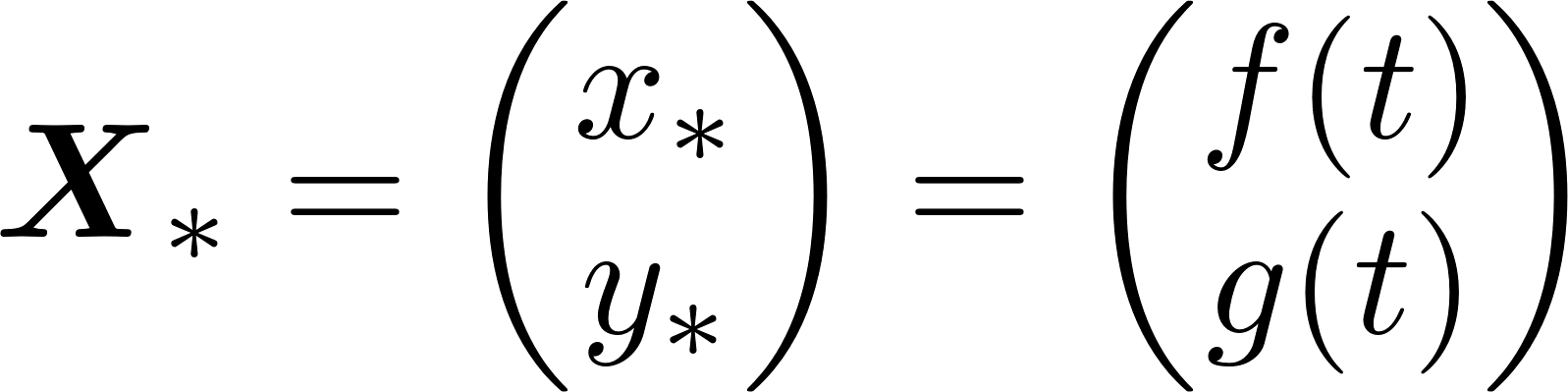
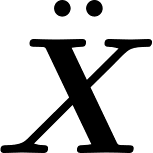
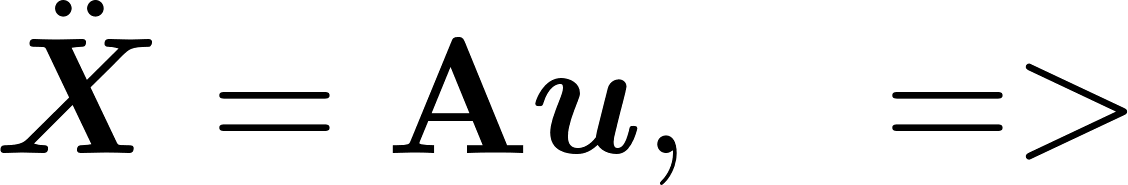
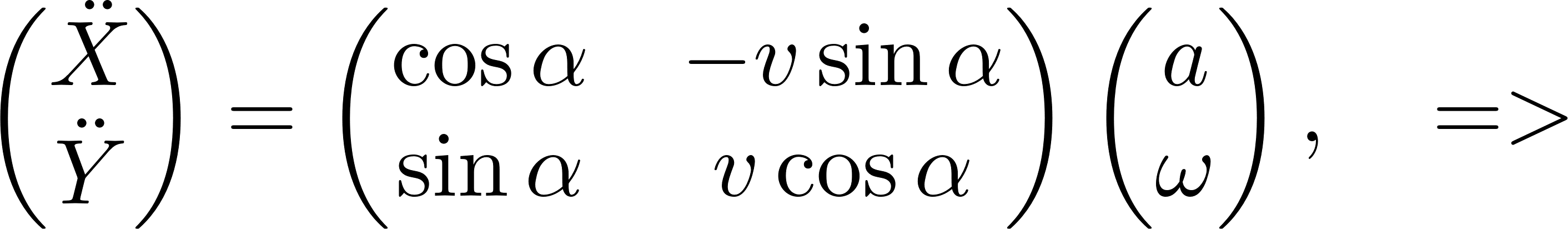
1. Выбираем шаг дискретизации . Чем меньше , тем больше точек будет на траектории. Либо задаем кол-во точек и рассчитываем относительно выбранного числа;
2. Вычисляем точки , подставляя t в функцию s();
3. Подставляем значения в заданные функции ;
4. Считаем линейную и угловую скорость;
5. Отправляем данные роботу. Движение. Коэффициент k;

## \documentclass{article} \usepackage{amsmath} \pagestyle{empty} \begin{document} \begin{gather*} \left\{ \begin{matrix} \ddot{X} = a\cos(\alpha) - v\omega\sin(\alpha), \\ \ddot{Y} = a\sin(\alpha) + v\omega\cos(\alpha), \\ \dot{\alpha} = \omega, \\ \dot{v} = a; \end{matrix} \right. \end{gather*} \end{document}\documentclass{article} \usepackage{amsmath} \pagestyle{empty} \begin{document} \begin{gather*} \left\{ \begin{matrix} \dot{X} = v\cos(\alpha), \\ \dot{Y} = v\sin(\alpha), \\ \dot{\alpha} = \omega. \end{matrix} \right. \end{gather*} \end{document}Формулы для движения с обратной связью

\documentclass{article}
\usepackage{amsmath}
\pagestyle{empty}
\newcommand{\bs}[1]{\boldsymbol{#1}}
\newcommand{\bm}[1]{\mathbf{#1}}
\begin{document}

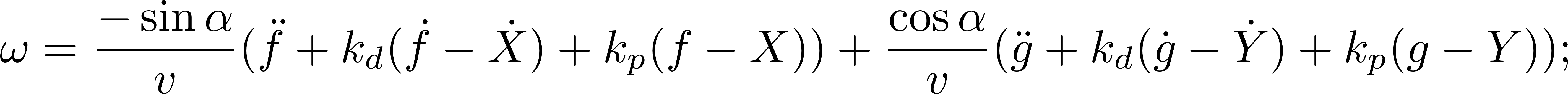
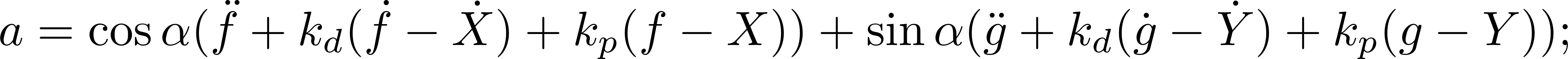
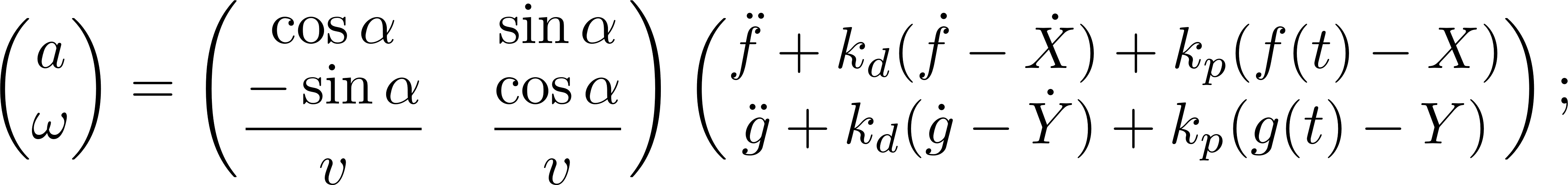
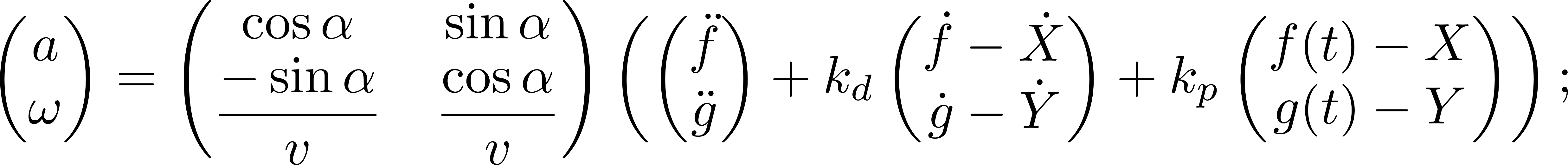
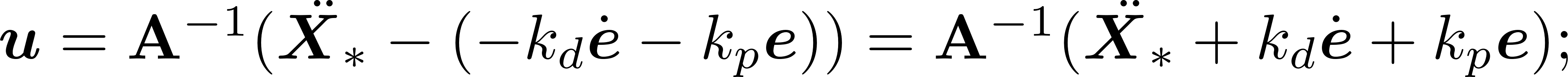
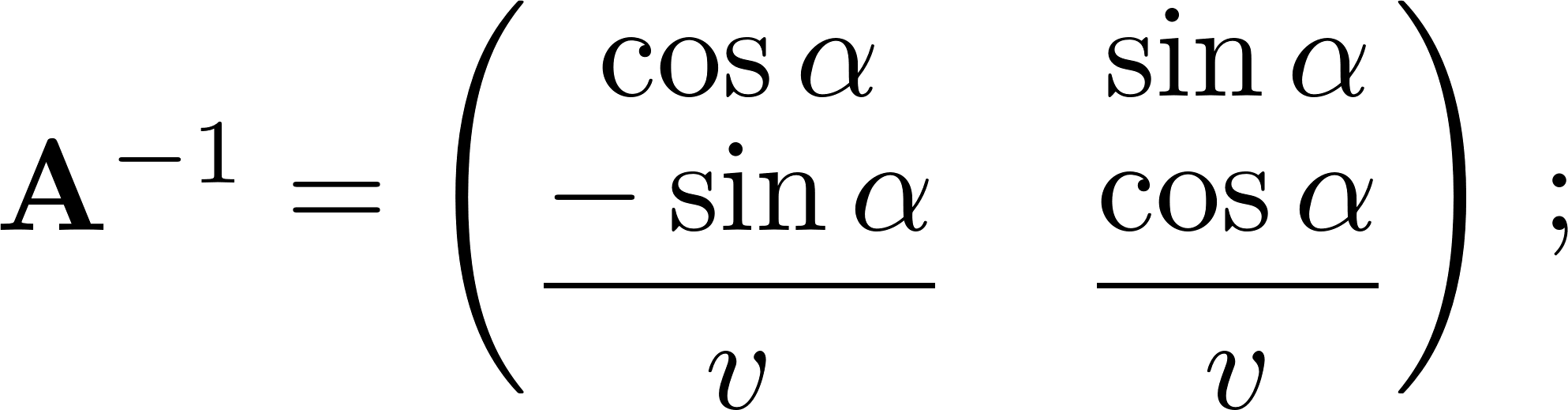
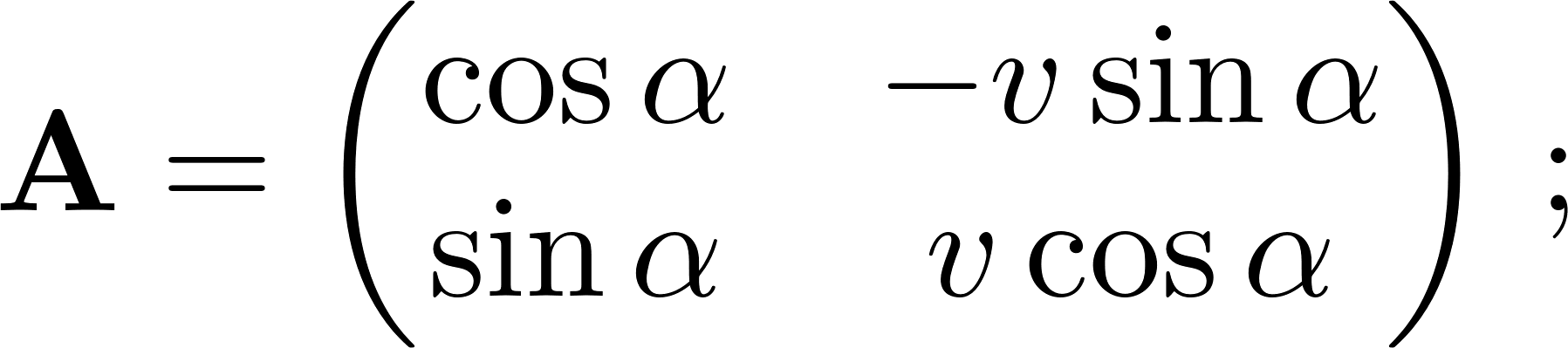
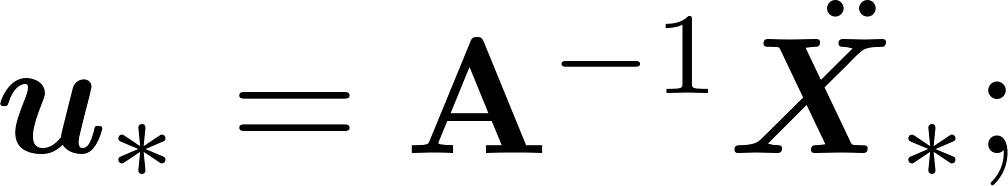
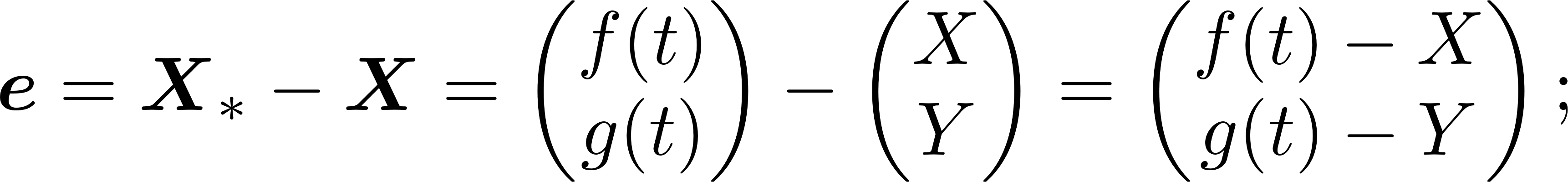
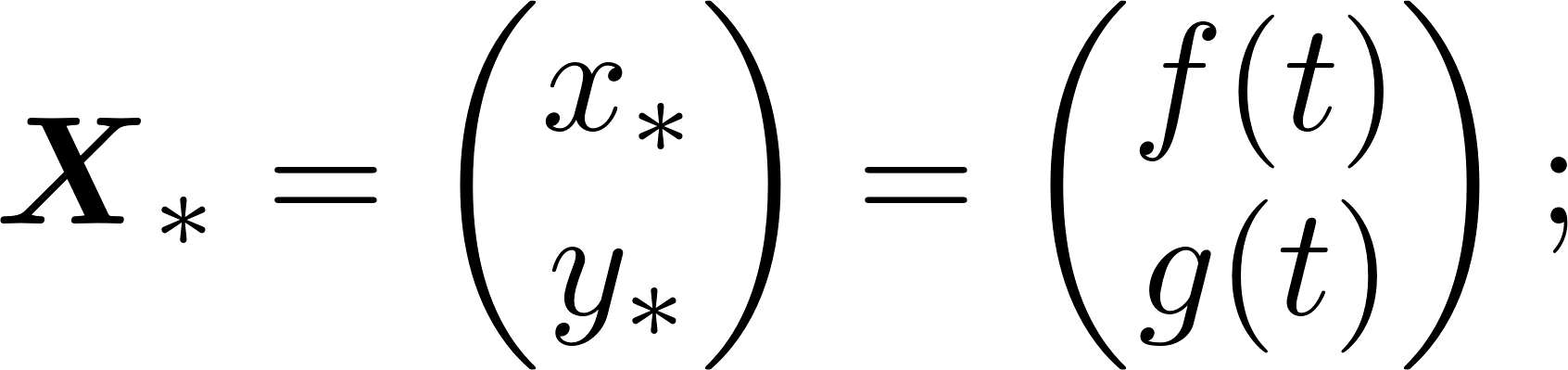
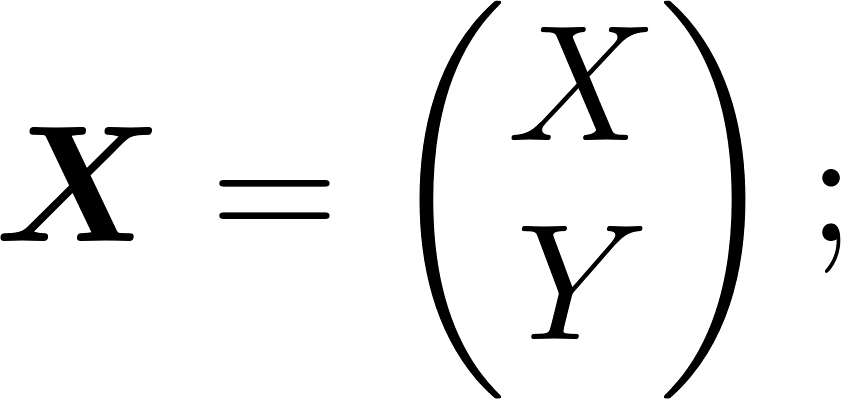
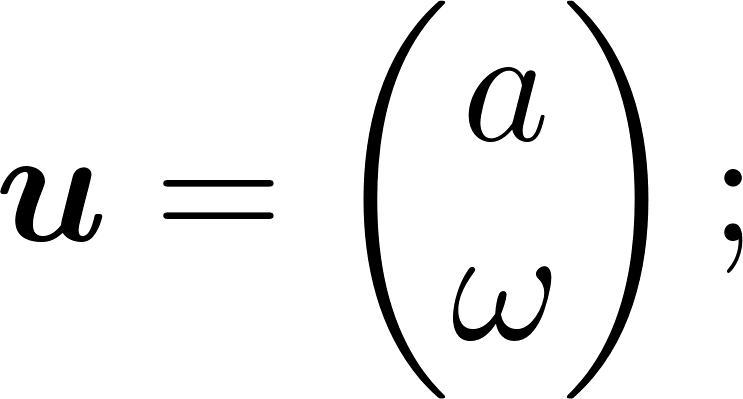
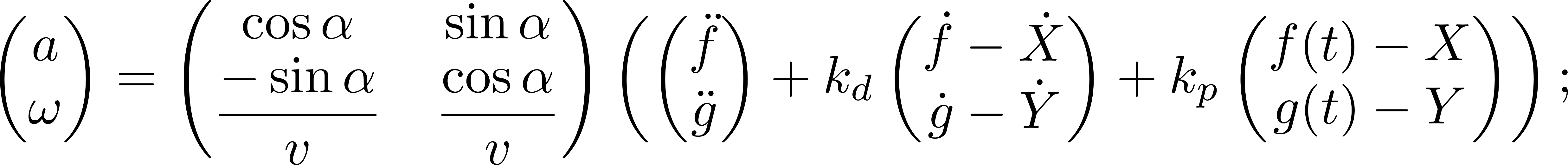
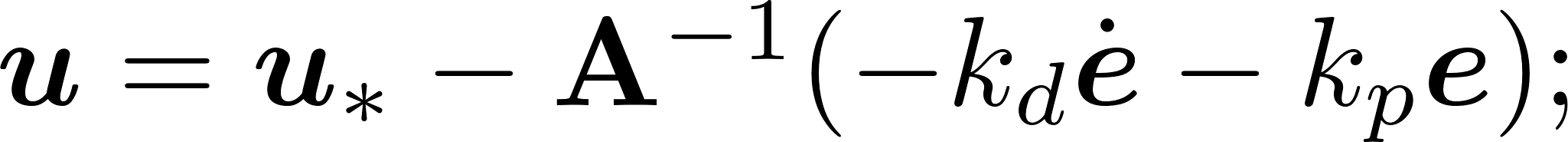
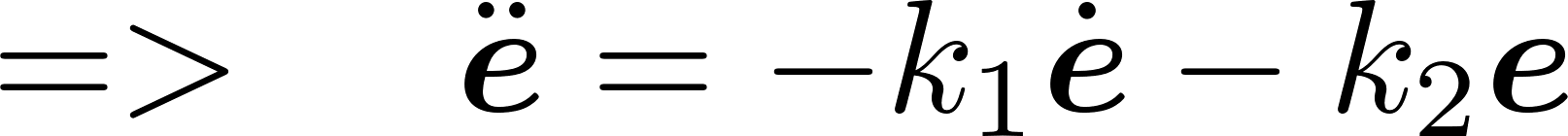
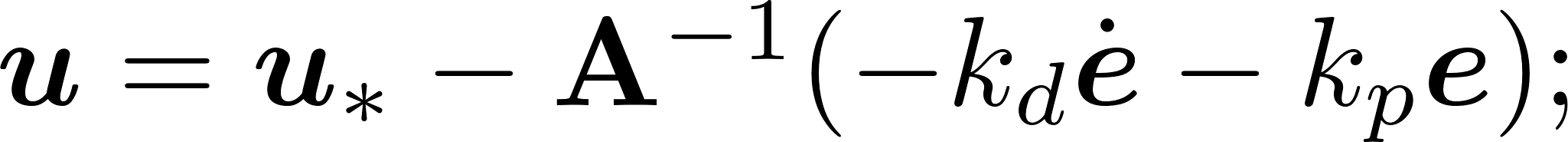
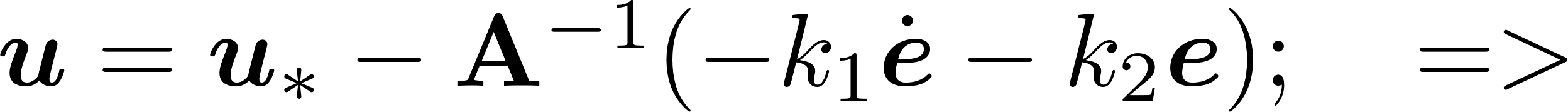
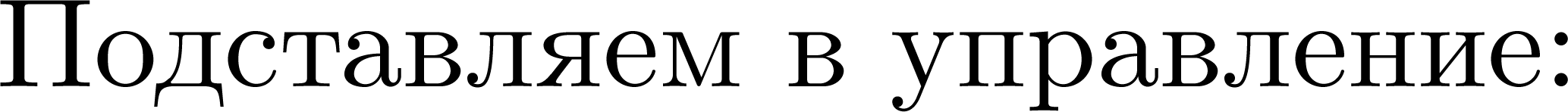
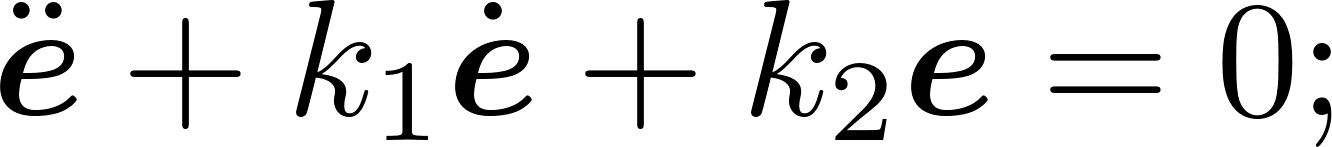
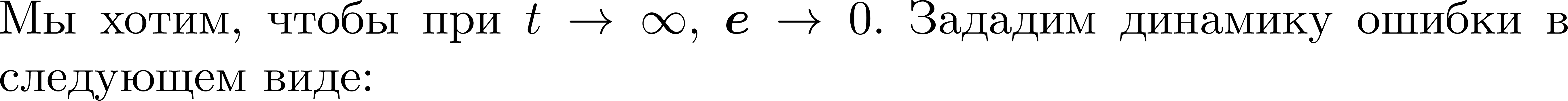
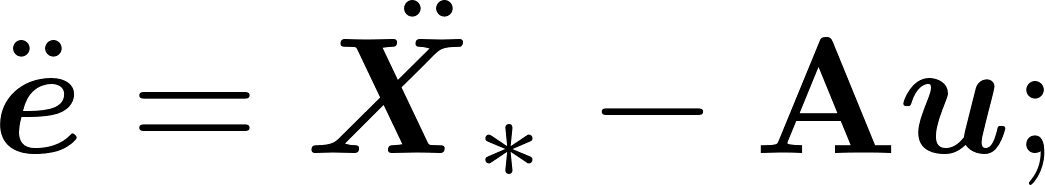
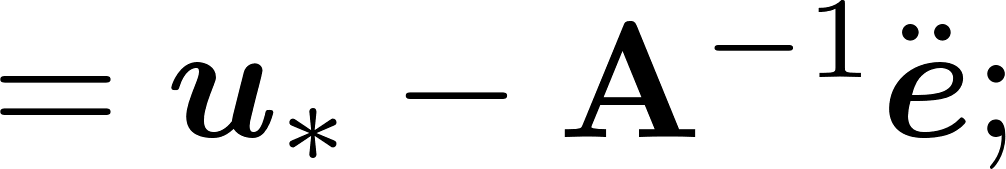
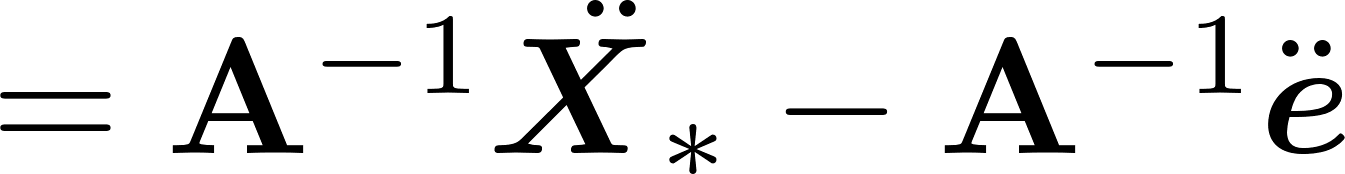
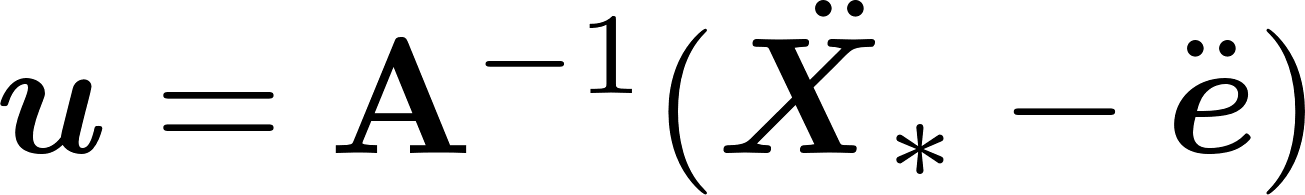
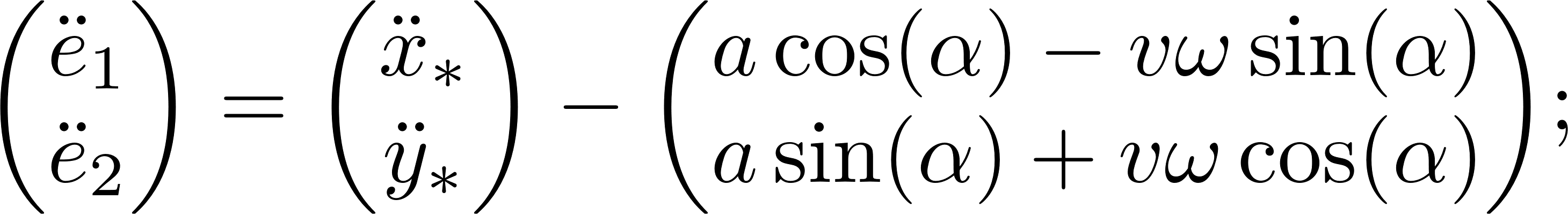
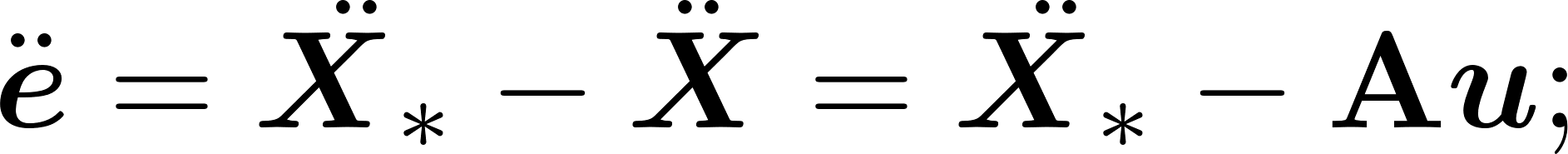
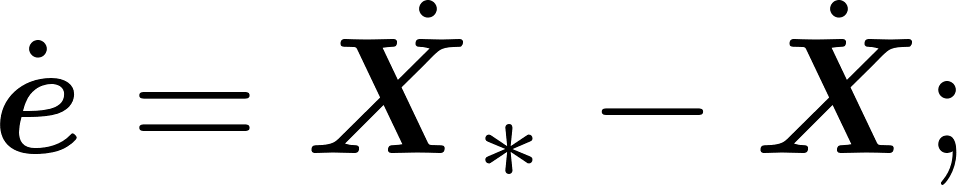
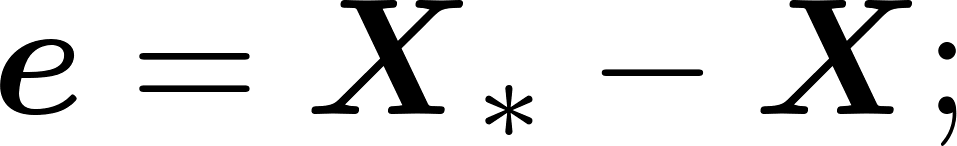
\begin{gather*}
\bs u = \bm A^{-1} \ddot{\bs X};
\end{gather*}


\end{document}



\documentclass{article}
\usepackage{amsmath}
\pagestyle{empty}
\usepackage[english,russian]{babel}
\usepackage[papersize={17 cm,15.4 cm}]{geometry}
\newcommand{\bs}[1]{\boldsymbol{#1}}
\newcommand{\bm}[1]{\mathbf{#1}}
\begin{document}
\noindent
$
\bs u_* = \bm A^{-1} \ddot{\bs X_*}
$ -- управляющее воздействия для движения по желаемой траектории -- feedforward control -- программное управление.

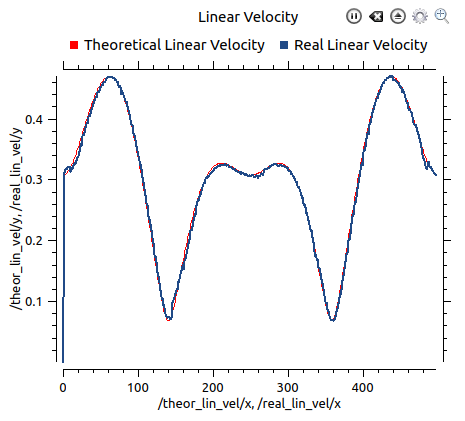

\end{document}

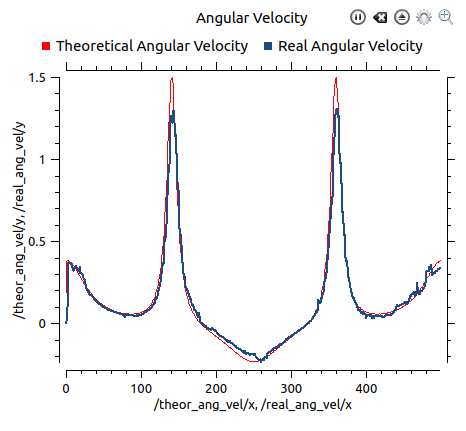


# Часть 2. Результаты

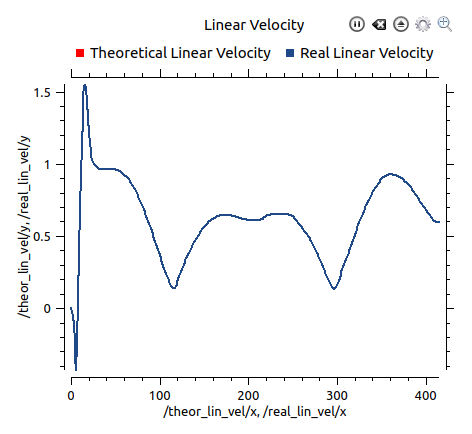
## Функция для реализации

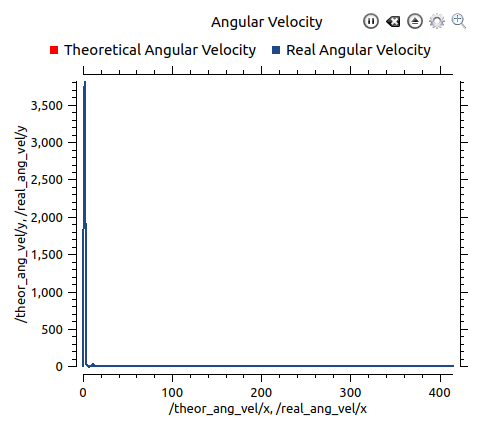
## Движение по траектории без обратной связи





## Движение по траектории с обратной связью





# Часть 3. Реализация кода программы

#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 = 0.3;

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) {

tfScalar yaw, pitch, roll;

tf::Quaternion q;

tf::quaternionMsgToTF(msg->pose.pose.orientation, q);

tf::Matrix3x3 m(q);

m.getEulerYPR(yaw, pitch, roll);

angleReal = yaw;

}

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;

}

float Xdt(float t)

{

return -time\_coef\*sin(t \* time\_coef) - 2\*time\_coef\*sin(2\*t \* time\_coef);

}

float Xdtdt(float t)

{

return -time\_coef\*time\_coef\*cos(t \* time\_coef) - 4\*time\_coef\*time\_coef\*cos(2\*t \* time\_coef);

}

float Y(float t)

{

return 2\*sin(t \* time\_coef);

}

float Ydt(float t)

{

return 2\*time\_coef\*cos(t \* time\_coef);

}

float Ydtdt(float t)

{

return -2\*time\_coef\*time\_coef\*sin(t \* time\_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);

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=%d, W=%d", velocity, omega);

}

mySigintHandler(1);

return 0;

}