**алгоритму управления траекторией двухмоторного манипулятора**

[1. Введение 1](#_Toc189045541)

[1.1. Значение точного управления в робототехнике 1](#_Toc189045542)

[1.1.1. Применение манипуляторов 1](#_Toc189045543)

[1.1.2. Проблемы, возникающие при управлении манипуляторами 2](#_Toc189045544)

[1.2. Роль манипуляторов в робототехнике 2](#_Toc189045545)

[1.2.1. Введение 2](#_Toc189045546)

[1.2.2. Определение и классификация манипуляторов 3](#_Toc189045547)

[1.2.2.1. Что такое манипулятор? 3](#_Toc189045548)

[1.2.2.2. Классификация манипуляторов 3](#_Toc189045549)

[1.2.3. Функции и применение манипуляторов 4](#_Toc189045550)

[1.2.3.1. Основные функции манипуляторов 4](#_Toc189045551)

[1.2.3.2. Промышленность 4](#_Toc189045552)

[1.2.3.3. Медицина 5](#_Toc189045553)

[1.2.3.4. Космос 5](#_Toc189045554)

[1.2.3.5. Логистика и складские системы 5](#_Toc189045555)

[1.3. Постановка задачи 5](#_Toc189045556)

[1.3.1. Введение в постановку задачи 5](#_Toc189045557)

[1.3.2. Цель работы 6](#_Toc189045558)

[1.3.3. Основные задачи 6](#_Toc189045559)

[1.3.3.1. Разработка математической модели манипулятора 6](#_Toc189045560)

[1.3.3.2. Исследование методов управления 6](#_Toc189045561)

[1.3.3.3. Разработка системы управления 6](#_Toc189045562)

[1.3.3.4. Реализация программного обеспечения 6](#_Toc189045563)

[1.3.3.5. Проведение экспериментов и тестирование 7](#_Toc189045564)

[1.3.4. Ограничения и допущения 7](#_Toc189045565)

[2. Структура системы управления 8](#_Toc189045566)

[2.1. Введение 8](#_Toc189045567)

[2.2. Общая структура системы управления 9](#_Toc189045568)

[2.3. Двухконтурная система управления 9](#_Toc189045569)

[2.3.1. Внешний контур (управление траекторией) 9](#_Toc189045570)

[2.3.2. Внутренний контур (позиционное управление) 10](#_Toc189045571)

[2.4. Контроллеры системы управления 10](#_Toc189045572)

[2.4.1. PID-контроллер 10](#_Toc189045573)

[2.4.2. LQR-контроллер 11](#_Toc189045574)

[2.5. LQR-управление 12](#_Toc189045575)

[2.6. Интеграция внешнего и внутреннего контуров 12](#_Toc189045576)

[3. Заключение 14](#_Toc189045577)

# Введение

В современных технологиях робототехнические манипуляторы играют ключевую роль в различных отраслях промышленности, медицины, логистики и даже космических исследованиях. Они используются для выполнения таких задач, как сборка деталей на конвейере, хирургические операции, сортировка грузов и работа в опасных условиях, где присутствие человека нежелательно.

Роботизированный манипулятор представляет собой механическую систему, состоящую из нескольких подвижных звеньев (звеньевых сегментов), соединенных суставами, которые приводятся в движение электродвигателями или другими приводами. Управление движением манипулятора требует сложных алгоритмов, которые обеспечивают **плавность траектории**, **точность позиционирования**, **эффективное распределение энергии** и **устойчивость к внешним возмущениям**.

В данной работе рассматривается **двухмоторный манипулятор**, использующий **пропорционально-интегрально-дифференциальный (PID) контроллер** и **линейно-квадратичный регулятор (LQR)** для обеспечения точности и стабильности управления. Эти алгоритмы позволяют минимизировать отклонения от заданной траектории и оптимизировать движение с точки зрения энергопотребления.

## Значение точного управления в робототехнике

### Применение манипуляторов

Двухмоторные манипуляторы применяются в следующих областях:

* **Промышленность**: роботизированные манипуляторы используются на сборочных линиях для выполнения операций, требующих высокой точности.
* **Медицина**: хирургические роботы, такие как Da Vinci, управляются с высокой точностью для выполнения сложных операций.
* **Логистика и складирование**: автоматические манипуляторы на складах перемещают товары, управляя их ориентацией.
* **Космические технологии**: роботизированные манипуляторы, например, Canadarm на Международной космической станции, выполняют задачи в условиях невесомости.
* **Автомобильная промышленность**: используются в производстве автомобилей для сварки, покраски и сборки.

### Проблемы, возникающие при управлении манипуляторами

Некоторые из сложностей, с которыми сталкиваются разработчики роботизированных манипуляторов:

* **Нелинейность динамической модели**: движение манипулятора описывается нелинейными дифференциальными уравнениями.
* **Ограниченные вычислительные мощности**: сложные алгоритмы требуют вычислений в реальном времени.
* **Внешние возмущения**: манипулятор может подвергаться воздействию случайных сил, таких как сопротивление воздуха или контакт с объектами.
* **Точность сенсоров**: даже небольшие ошибки в измерении углов могут привести к значительным отклонениям.

## Роль манипуляторов в робототехнике

### Введение

Робототехника является одной из самых динамично развивающихся областей науки и техники. Одним из ключевых элементов современных роботов являются **манипуляторы** – механические устройства, которые позволяют роботам взаимодействовать с окружающей средой. Манипуляторы используются в промышленности, медицине, авиации, космосе и многих других сферах, где требуется высокая точность, автоматизация и надежность выполнения задач.

Данный раздел подробно рассматривает:

* Определение и классификацию манипуляторов;
* Их функции и применение в различных отраслях;
* Преимущества и ограничения использования роботизированных манипуляторов;
* Основные технологические тенденции в разработке манипуляторов.

### Определение и классификация манипуляторов

#### Что такое манипулятор?

**Манипулятор** – это механическая система, состоящая из последовательности звеньев, соединенных подвижными суставами, которая может выполнять заданные движения для манипуляции объектами. Манипуляторы могут быть как автономными, так и управляемыми человеком.

#### Классификация манипуляторов

Существует несколько способов классификации манипуляторов:

1. **По степени автоматизации:**
   * **Автоматизированные** – работают по заранее заданным программам.
   * **Дистанционно управляемые** – управляются оператором в реальном времени.
   * **Гибридные** – комбинируют автоматизированное управление с возможностью вмешательства оператора.
2. **По числу степеней свободы:**
   * **Простые** – имеют 2-3 степени свободы (например, манипуляторы с линейными перемещениями).
   * **Многофункциональные** – имеют 6 и более степеней свободы (используются в сборке, сварке, хирургии).
3. **По типу привода:**
   * **Электрические** – наиболее распространенные, отличаются высокой точностью и низким уровнем шума.
   * **Гидравлические** – обладают высокой мощностью, применяются в тяжелых промышленных роботах.
   * **Пневматические** – используются в легких и быстрых манипуляторах.
4. **По области применения:**
   * **Промышленные** – используются в производстве и сборке.
   * **Медицинские** – применяются в хирургии и реабилитации.
   * **Космические** – например, Canadarm на Международной космической станции.
   * **Логистические** – используются в складах для автоматизированного перемещения товаров.

### Функции и применение манипуляторов

#### Основные функции манипуляторов

Манипуляторы выполняют широкий спектр задач, включая:

* **Перемещение объектов** – от простых грузоподъемных операций до точного позиционирования.
* **Сборку** – монтаж компонентов в точных последовательностях.
* **Операции с высокой точностью** – сварка, лазерная обработка, 3D-печать.
* **Манипуляции в агрессивных средах** – работа с химическими веществами, радиацией.
* **Медицинские манипуляции** – микрохирургические операции.
* **3.2 Применение манипуляторов в различных отраслях**

#### Промышленность

Промышленные манипуляторы являются неотъемлемой частью автоматизированных производственных процессов. Их основные задачи:

* Сварка и покраска автомобилей.
* Сборка электроники (пайка, тестирование).
* Обслуживание станков и конвейерных линий.

#### Медицина

Роботизированные манипуляторы используются в медицине для выполнения сложных операций с высокой точностью. Примеры:

* **Система Da Vinci** – управляемый хирургический манипулятор для минимально инвазивных операций.
* **Реабилитационные манипуляторы** – помогают пациентам восстанавливать двигательные функции.

#### Космос

В космической отрасли манипуляторы применяются для работы в условиях невесомости:

* **Канадарм** – манипулятор, используемый на МКС для захвата грузовых кораблей.
* **Роботизированные манипуляторы марсоходов** – для сбора образцов грунта.

#### Логистика и складские системы

Манипуляторы используются в логистике и складах для автоматического перемещения товаров:

* **Роботизированные сортировщики** – автоматическое распределение посылок.
* **Автономные погрузчики** – перемещение паллет и коробок.

## Постановка задачи

### Введение в постановку задачи

Автоматизированные манипуляторы являются важной частью **современных робототехнических систем**. Они обеспечивают выполнение сложных операций с высокой точностью, скоростью и стабильностью. Для эффективной работы таких систем требуется **разработка алгоритмов управления**, обеспечивающих **плавность траекторий**, **минимизацию ошибок** и **оптимизацию энергозатрат**.

В данной работе рассматривается **двухмоторный манипулятор**, который управляется с использованием **PID-контроллера** и **LQR-регулятора**. Основная цель — обеспечить **плавное и точное движение манипулятора**, минимизируя ошибки и контролируя энергопотребление.

### Цель работы

Целью данной работы является разработка алгоритма управления двухмоторным манипулятором, обеспечивающего точное следование траектории, минимизацию ошибки, устойчивость системы и оптимальное энергопотребление.

### Основные задачи

Для достижения поставленной цели необходимо выполнить следующие задачи:

#### Разработка математической модели манипулятора

* Определение кинематических уравнений, описывающих положение конечного звена манипулятора.
* Построение динамической модели, описывающей движение моторов и взаимодействие сил.
* Анализ степеней свободы и их влияния на управление манипулятором.

#### Исследование методов управления

* Изучение классических методов управления (PID-регулятор) и их применения к двухмоторным манипуляторам.
* Анализ оптимального управления с помощью линейно-квадратичного регулятора (LQR).
* Определение ограничений скорости, ускорения и управления.

#### Разработка системы управления

* Внешний контур управления: расчет траекторий движения манипулятора.
* Внутренний контур управления: стабилизация углов с помощью PID и LQR.
* Интеграция двухконтурного управления для обеспечения точности.

#### Реализация программного обеспечения

* Разработка программного кода на C++ для управления манипулятором.
* Использование библиотек SFML для визуализации движений манипулятора.
* Реализация алгоритмов управления в реальном времени.

#### Проведение экспериментов и тестирование

* Тестирование системы управления в различных режимах работы.
* Анализ ошибок и их влияние на точность управления.
* Оценка устойчивости системы при изменении параметров.

### Ограничения и допущения

При выполнении работы необходимо учитывать следующие ограничения:

1. Физические ограничения манипулятора – максимальные углы поворота, пределы скорости и ускорения моторов.
2. Аппаратные ограничения – система будет моделироваться в программном симуляторе, а не на реальном устройстве.
3. Внешние воздействия – модель не учитывает динамику влияния внешних сил (например, трения, воздушного сопротивления).
4. Точность измерений – используются идеальные данные о положении моторов без учета погрешностей сенсоров.

# Структура системы управления

## Введение

Система управления двухмоторным манипулятором представляет собой комплекс аппаратных и программных решений, направленных на обеспечение **точного, плавного и оптимизированного движения** манипулятора в соответствии с заданной траекторией.

Основной задачей системы управления является **обеспечение точного позиционирования** каждого из звеньев манипулятора с учетом динамических характеристик, ограничений привода и заданной траектории движения.

Система управления двухмоторным манипулятором включает в себя:

* **Контур траекторного управления (внешний контур)** – определяет траекторию движения и формирует команду управления.
* **Контур позиционного управления (внутренний контур)** – стабилизирует движение, компенсирует ошибки и корректирует углы моторов в реальном времени.
* **Систему обратной связи** – позволяет корректировать положение манипулятора на основе измеренных данных.
* **Контроллеры (PID и LQR)** – управляют приводами манипулятора

## Общая структура системы управления

Система управления состоит из следующих основных компонентов:

1. **Генератор траектории** – рассчитывает целевые углы моторов на каждом шаге движения.
2. **Контроллеры (PID и LQR)** – вычисляют корректирующие сигналы для привода.
3. **Приводы (моторы)** – обеспечивают движение манипулятора.
4. **Датчики обратной связи** – измеряют текущие углы поворота звеньев.
5. **Модуль визуализации (SFML)** – позволяет отслеживать движение манипулятора в реальном времени.

## Двухконтурная система управления

### Внешний контур (управление траекторией)

**Функции внешнего контура:**

* Определение требуемого положения манипулятора в каждый момент времени.
* Генерация целевых углов звеньев на каждом шаге движения.
* Обеспечение плавности движения за счет создания **оптимизированной траектории**.

**Алгоритм работы внешнего контура:**

1. Получение начальной и конечной точки движения.
2. Расчет промежуточных углов поворота с учетом ограничений.
3. Передача целевых значений в контур позиционного управления.

**Пример расчета линейной траектории:**

std::vector<double> calculateTargetTrajectory(double start\_pos, double end\_pos, int num\_points) {  
 std::vector<double> trajectory;  
 double step = (end\_pos - start\_pos) / (num\_points - 1);  
 for (int i = 0; i < num\_points; ++i) {  
 trajectory.push\_back(start\_pos + i \* step);  
 }  
 return trajectory;  
}

### Внутренний контур (позиционное управление)

**Функции внутреннего контура:**

* Регулирование положения звеньев манипулятора.
* Минимизация ошибки угла с использованием **PID и LQR контроллеров**.
* Обеспечение устойчивости системы.

**Алгоритм работы внутреннего контура:**

1. Измерение текущего положения манипулятора.
2. Вычисление ошибки относительно целевого угла.
3. Формирование управляющего воздействия с помощью PID и LQR.
4. Коррекция движения манипулятора.

## Контроллеры системы управления

### PID-контроллер

**Формула управления:**

где:

— коэффициенты PID-контроллера

class PIDController {  
 double target\_angle = 21;  
 double last\_error = 0;  
 double integral = 0;  
public:  
 void setTarget(double target) { target\_angle = target; }  
 double computeControl(double current\_angle, float dt) {  
 double error = target\_angle - current\_angle;  
 integral += error \* dt;  
 double derivative = (error - last\_error) / dt;  
 last\_error = error;  
 return 0.8 \* error + 0.03 \* integral + 0.1 \* derivative;  
 }  
};

Положение конечного звена манипулятора определяется уравнениями:

где:

* и — длины звеньев манипулятора.

и — углы моторов

### LQR-контроллер

**Функция затрат:**

## LQR-управление

LQR-контроллер минимизирует функцию затрат:

где:

* отвечает за минимизацию отклонения от цели.
* минимизирует энергетические затраты.

Оптимальное управление:

где решается через **уравнение Риккати**:

LQRController {  
 Eigen::MatrixXd K;  
public:  
 LQRController() {  
 Eigen::MatrixXd A(1, 1), B(1, 1), Q(1, 1), R(1, 1), P(1, 1);  
 A << 1.0;  
 B << 1.0;  
 Q << 1.0;  
 R << 0.5;  
 P = Q;  
 for (int i = 0; i < 100; ++i) {  
 P = Q + A.transpose() \* P \* A - A.transpose() \* P \* B \* (R + B.transpose() \* P \* B).inverse() \* B.transpose() \* P \* A;  
 }  
 K = (R + B.transpose() \* P \* B).inverse() \* B.transpose() \* P \* A;  
 }  
 double computeControl(double state) {  
 return -K(0, 0) \* state;  
 }  
};

## Интеграция внешнего и внутреннего контуров

**Основной цикл управления:**

int main() {

sf::RenderWindow window(sf::VideoMode(800, 600), "Two-Motor Manipulator");

Motor motorA, motorB;

PIDController pidControllerA, pidControllerB;

LQRController lqrControllerA, lqrControllerB;

std::vector<double> trajectory = calculateTargetTrajectory(45, 21, 10);

int step = 0;

sf::Clock clock;

while (window.isOpen()) {

sf::Time elapsed = clock.restart();

float frameTime = std::max(elapsed.asSeconds(), 0.0001f);

double target\_angle = trajectory[std::min(step, (int)trajectory.size() - 1)];

double control\_signalA = pidControllerA.computeControl(motorA.angle, frameTime)

+ lqrControllerA.computeControl(motorA.angle - target\_angle);

motorA.update(control\_signalA, frameTime);

double control\_signalB = pidControllerB.computeControl(motorB.angle, frameTime)

+ lqrControllerB.computeControl(motorB.angle - target\_angle);

motorB.update(control\_signalB, frameTime);

step++;

}

return 0;

}

# Заключение

**Отчет охватывает:**

* Математическое моделирование
* Реализацию PID и LQR-контроллеров
* Генерацию траектории
* Явные внутренний и внешний контуры управления

# Ссылки

* <https://www.sfml-dev.org/tutorials/3.0/getting-started/visual-studio/#creating-and-configuring-an-sfml-project>
* <https://eigen.tuxfamily.org/dox/namespaceEigen.html>

# приложение

## код

#define \_USE\_MATH\_DEFINES

#include <SFML/Graphics.hpp>

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

#include <string>

#include <Eigen/Dense> // For solving Riccati equation

// Constants

const double KP = 0.8;

const double KI = 0.03;

const double KD = 0.1;

const double LQR\_Q = 1.0;

const double LQR\_R = 0.5;

const double MAX\_VELOCITY = 3.0;

const float MIN\_DT = 0.0001;

constexpr double DEG2RAD = M\_PI / 180.0;

const float ARM\_LENGTH1 = 150.f;

const float ARM\_LENGTH2 = 100.f;

const int WINDOW\_WIDTH = 800;

const int WINDOW\_HEIGHT = 600;

// Motor Structure

struct Motor {

double angle = 45;

double velocity = 0;

double integral\_error = 0;

void update(double control\_signal, float dt) {

dt = std::max(static\_cast<float>(dt), MIN\_DT);

velocity = std::clamp(control\_signal, -MAX\_VELOCITY, MAX\_VELOCITY);

angle = std::clamp(angle + velocity \* dt, 10.0, 170.0);

}

};

// Function Declarations

std::vector<double> calculateTargetTrajectory(double start\_pos, double end\_pos, int num\_points);

double measureCurrentPosition(const Motor& motor);

double calculateError(double target\_pos, double current\_pos);

double applyPIDController(double error, double integral, double derivative);

double applyLQRController(double state\_vector, double control\_input);

void initializeSystem();

// PID Controller

class PIDController {

double target\_angle = 21;

double last\_error = 0;

double integral = 0;

public:

void setTarget(double target) { target\_angle = target; }

double computeControl(double current\_angle, float dt) {

dt = std::max(static\_cast<float>(dt), MIN\_DT);

double error = target\_angle - current\_angle;

integral += error \* dt;

double derivative = (error - last\_error) / dt;

last\_error = error;

return KP \* error + KI \* integral + KD \* derivative;

}

};

// LQR Контроллер

class LQRController {

Eigen::MatrixXd K;

public:

LQRController() {

Eigen::MatrixXd A(1, 1), B(1, 1), Q(1, 1), R(1, 1), P(1, 1);

A << 1.0;

B << 1.0;

Q << LQR\_Q;

R << LQR\_R;

// Решение уравнения Риккатти итеративным методом

P = Q;

for (int i = 0; i < 100; ++i) {

P = Q + A.transpose() \* P \* A - A.transpose() \* P \* B \* (R + B.transpose() \* P \* B).inverse() \* B.transpose() \* P \* A;

}

K = (R + B.transpose() \* P \* B).inverse() \* B.transpose() \* P \* A;

}

double computeControl(double state) {

return -K(0, 0) \* state;

}

};

void initializeSystem() {

std::cout << "Initializing system..." << std::endl;

std::cout << "PID Constants: KP=" << KP << " KI=" << KI << " KD=" << KD << std::endl;

std::cout << "LQR Constants: Q=" << LQR\_Q << " R=" << LQR\_R << std::endl;

std::cout << "Motion Constraints: MAX\_VELOCITY=" << MAX\_VELOCITY << " MIN\_DT=" << MIN\_DT << std::endl;

std::cout << "Manipulator Arm: Length1=" << ARM\_LENGTH1 << " Length2=" << ARM\_LENGTH2 << std::endl;

std::cout << "Window Size: WIDTH=" << WINDOW\_WIDTH << " HEIGHT=" << WINDOW\_HEIGHT << std::endl;

}

std::vector<double> calculateTargetTrajectory(double start\_pos, double end\_pos, int num\_points) {

std::vector<double> trajectory;

double step = (end\_pos - start\_pos) / (num\_points - 1);

for (int i = 0; i < num\_points; ++i) {

trajectory.push\_back(start\_pos + i \* step);

}

return trajectory;

}

double measureCurrentPosition(const Motor& motor) {

return motor.angle;

}

double calculateError(double target\_pos, double current\_pos) {

return target\_pos - current\_pos;

}

double applyPIDController(double error, double integral, double derivative) {

return KP \* error + KI \* integral + KD \* derivative;

}

double applyLQRController(double state\_vector, double control\_input) {

return -LQR\_Q / (LQR\_Q + LQR\_R) \* state\_vector;

}

int main() {

initializeSystem();

sf::RenderWindow window(sf::VideoMode({ WINDOW\_WIDTH, WINDOW\_HEIGHT }), "Two-Motor Manipulator");

Motor motorA, motorB;

PIDController pidControllerA, pidControllerB;

LQRController lqrControllerA, lqrControllerB;

std::vector<double> trajectory = calculateTargetTrajectory(45, 21, 10);

double dt = 0.1;

int step = 0;

sf::Clock clock;

// Create shapes for the arms

sf::RectangleShape arm1(sf::Vector2f(ARM\_LENGTH1, 10));

arm1.setFillColor(sf::Color::Red);

arm1.setOrigin({ 0, 5 });

sf::RectangleShape arm2(sf::Vector2f(ARM\_LENGTH2, 10));

arm2.setFillColor(sf::Color::Blue);

arm2.setOrigin({ 0, 5 });

while (window.isOpen()) {

sf::Time elapsed = clock.restart();

float frameTime = std::max(elapsed.asSeconds(), MIN\_DT);

while (const std::optional event = window.pollEvent()) {

if (event->is<sf::Event::Closed>()) {

window.close();

}

}

double target\_angle = trajectory[std::min(step, (int)trajectory.size() - 1)];

double current\_posA = measureCurrentPosition(motorA);

double current\_posB = measureCurrentPosition(motorB);

double errorA = calculateError(target\_angle, current\_posA);

double errorB = calculateError(target\_angle, current\_posB);

double control\_signalA = pidControllerA.computeControl(motorA.angle, frameTime) + lqrControllerA.computeControl(motorA.angle - target\_angle);

double control\_signalB = pidControllerB.computeControl(motorB.angle, frameTime) + lqrControllerB.computeControl(motorB.angle - target\_angle);

motorA.update(control\_signalA, frameTime);

motorB.update(control\_signalB, frameTime);

if (step % 10 == 0) {

std::cout << "Motor1 Angle: " << motorA.angle << " | Motor2 Angle: " << motorB.angle << std::endl;

}

step++;

sf::Vector2f basePosition(WINDOW\_WIDTH / 2.f, WINDOW\_HEIGHT / 2.f);

// Update arm positions

arm1.setPosition(basePosition);

arm1.setRotation(sf::degrees(motorA.angle));

sf::Vector2f arm1End(WINDOW\_WIDTH / 2 + ARM\_LENGTH1 \* std::cos(motorA.angle \* DEG2RAD),

WINDOW\_HEIGHT / 2 + ARM\_LENGTH1 \* std::sin(motorA.angle \* DEG2RAD));

arm2.setPosition(arm1End);

arm2.setRotation(sf::degrees(motorA.angle + motorB.angle));

// Clear the window

window.clear();

// Draw the arms

window.draw(arm1);

window.draw(arm2);

// Display the window

window.display();

}

return 0;

}

## результаты









