Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _СПЕЦИАЛЬНОЕ МАШИНОСТРОЕНИЕ_____

КАФЕДРА _ _РОБОТОТЕХНИЧЕСКИЕ СИСТЕМЫ И МЕХАТРОНИКА_____

ОТЧЕТ О УПРАВЛЕНИИ СКОРОСТИ

Студент ____СМ7И-32М__ _____18.10.2023___Гунъе Цзиньлун__
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____18.10.2023___Вассуф Язан__
(Подпись, дата) (И.О.Фамилия)

2023 г.

Оглавление

1. Постановка задачи

1. Задача реализовать и настроить контроллер управления скоростью на базе модуля velocity_controller. Код этого модуля содержится в файле [velocity_controller/src/velocity_controller.cpp](https://github.com/AndreyMinin/MobileRobots/blob/master/mr_ws/src/velocity_controller/src/velocity_controller.cpp). Код реализован в виде фукций-колбеков, котрые подписаны на необходимые ros сообщения (заданную скорость, текущую скорость, таймер). Необходимо отредактировать код, реализовав контроллер управления скоростью. Для проверки собранного модуля выполнить пункты 4,5 (саму модель можно не перезапускать)

2. Добавим реалистичности: шум в данные одометрии. Для этого нужно запустить модель с помощью команды

```bash
roslaunch velocity_controller throttle_cart.launch noise:=0.2
```

где 0.2 - значение шума одометрии.

После запуска модели с шумом необходимо и проверить работоспособность разработанного контроллера в присутствии шумов. Доработать контроллер скорости, добавив сглаживание на входные данные.

2. Контроллер управления скоростью без фильтра

Принцип алгоритма

В задаче 1 требуется разработать контроллер, который позволит мобильному роботу двигаться со скоростью, соответствующей заданной (ускорение - постоянная скорость - замедление). Исходные данные о скорости получаются из одометрии мобильного робота без шумового воздействия. В данном случае я использую алгоритм дискретного PID-регулирования для отслеживания скорости. Его математическое выражение представлено ниже:

$$u(k) = k_p * e(k) + k_i * \sum_{i=0}^{k} e(i) + k_d * [e(k) - e(k-1)]$$

Где $e(k) = V_{\text{цель}} - V_{\text{факт}}$, ошибка скорости в момент времени k.

$u(k)$-значение газа/тормоза.

k_p, k_i, k_dпредставляют собой пропорциональный, интегральный и дифференциальный коэффициенты соответственно.

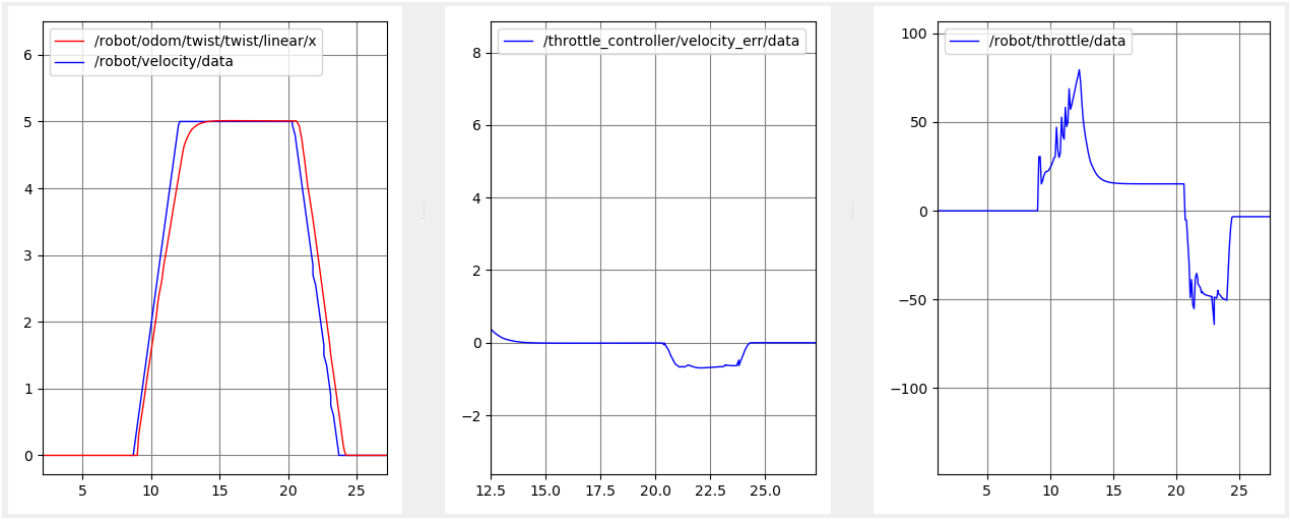
Код:

```
double last_error = desired_velocity - current_velocity ;
double error_integral;
void on_timer(const ros::TimerEvent& event) {
double p_factor = 100;
double d_factor = 0.1;
double i_factor = 1.1;
auto t = ros::Time::now();
auto dt = (t - last_timer_time).toSec();
last_timer_time = t;
std_msgs::Float32 throttle_cmd;
// place code here to calculate throttle/brake
double error = desired_velocity - current_velocity;
double diff_err = error - last_error;
last_error = error;
error_integral += error;
double throttle = p_factor * error
+ d_factor * diff_err
+ i_factor * error_integral;
throttle_cmd.data = throttle;
throttle_pub.publish(throttle_cmd);
}
```

Объяснение:

Здесь объявляются необходимые переменные, включая ошибку скорости во время k-1, сумму интегралов ошибок и параметр pid. Рассчитайте необходимые значения газа/тормоза на основе дискретной формулы ПИД.

Эффект управления:



Видно, что этот контроллер может заставить мобильного робота быстро достичь целевой скорости, а общая ошибка невелика.

3. Контроллер управления скоростью с фильтром

Принцип алгоритма

Во второй задаче информация о скорости, считываемая с одометра, подвергается помехам, поэтому необходимо использовать фильтр для максимально возможной фильтрации шума, а затем отфильтрованная скорость используется для управления мобильным роботом.

Здесь используется фильтр Гаусса. Принцип состоит в том, чтобы установить окно фильтра, вычислить средневзвешенное значение всех входных данных в окне в соответствии с функцией распределения Гаусса и передать среднее значение контроллеру как реальное значение входных данных в момент времени k. В момент времени k+1 первые данные в окне удаляются, а новые входные данные добавляются в окно для реализации перемещения окна.

Код

```
double last_error = desired_velocity - current_velocity;
double error_integral;
std::vector<double> velocity_history;
const int window_size = 18; // size of window of moving average filter

// Define Gaussian function
double gaussian(double x, double mean, double stddev) {
return exp(-0.5 * pow((x - mean) / stddev, 2)) / (stddev * sqrt(2 * M_PI));
}

// Calculate weight and weighted average
for (int i = -window_size/2; i <= window_size/2; ++i) {
if (index + i >= 0 && index + i < data.size()) {
double weight = gaussian(i, 0, stddev);
result += data[index + i] * weight;
total_weight += weight;
}
}

return result / total_weight;
}

void on_timer(const ros::TimerEvent& event) {
double p_factor = 50;
double d_factor = 0.1;
double i_factor = 0;

auto t = ros::Time::now();
auto dt = (t - last_timer_time).toSec();
last_timer_time = t;
```

```
std_msgs::Float32 throttle_cmd;

// place code here to calculate throttle/brake
velocity_history.push_back(current_velocity);

// use moving average filter to processes the velocity
// delete the earliest value if windows is full
if (velocity_history.size() > window_size) {
velocity_history.erase(velocity_history.begin());
}

//calculate the average velocity in this windows
for (double v : velocity_history) {
filtered_current_velocity += v;
}
filtered_current_velocity /= velocity_history.size();

// Set standard deviation
double stddev = 10;
double filtered_current_velocity = gaussian_filter(velocity_history, velocity_history.size()-1, window_size,
stddev);

double error = desired_velocity - filtered_current_velocity;
// double error = desired_velocity - current_velocity;
double diff_err = error - last_error;
last_error = error;
error_integral += error;

double throttle = p_factor * error
+ d_factor * diff_err
+ i_factor * error_integral;

throttle_cmd.data = throttle;
throttle_pub.publish(throttle_cmd);
}
```

Объяснение:

На основе дискретного ПИД-алгоритма добавлен фильтр Гаусса. Вектор Velocity_history определен для хранения значения скорости, а Window_size используется для установки размера окна. В функции обратного вызова убедитесь, что длина Velocity_history соответствует пределу window_size. Затем введите значение скорости в Velocity_history в функцию фильтра Гаусса, чтобы найти средневзвешенное значение в качестве результата фильтрации.

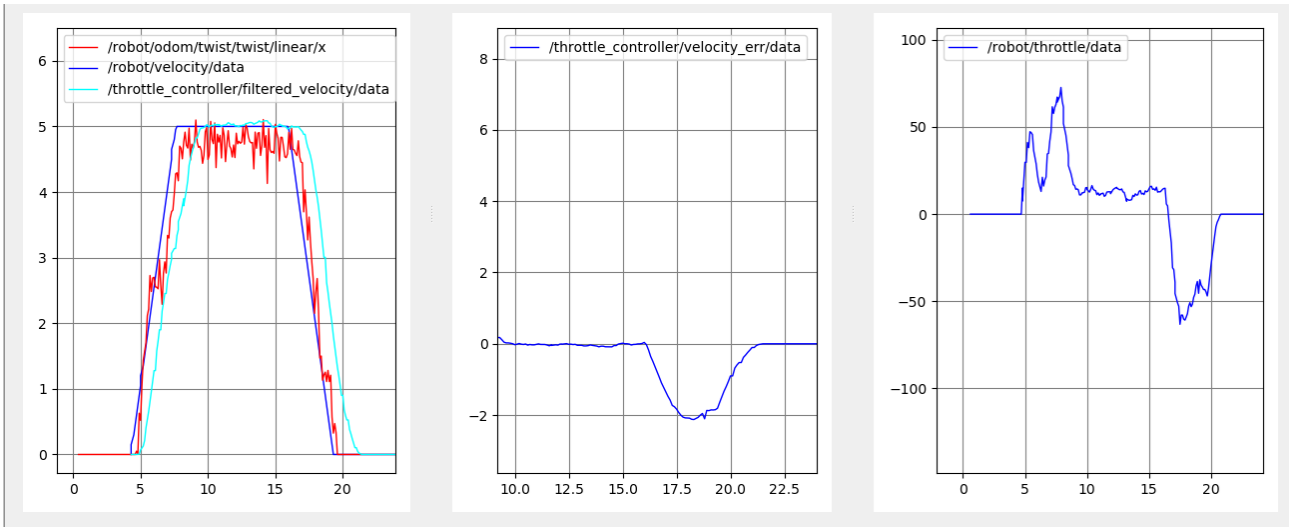
```
void on_command_velocity(const std_msgs::Float32& msg) {
desired_velocity = msg.data;
std_msgs::Float32 err;
// err.data = desired_velocity - current_velocity;
err.data = desired_velocity - filtered_current_velocity; // change the error to error between the desired
and the filtered
```

```
// ROS_INFO_STREAM_COND(cmd_velocity > 0.1, "current velocity " << current_velocity << " err = " << err.data);
err_pub.publish(err);
std_msgs::Float32 filtered_v;
filtered_v.data = filtered_current_velocity;
filtered_pub.publish(filtered_v);
}
```

Объяснение:

Чтобы облегчить наблюдение за эффектом фильтрации, отфильтрованная информация о скорости публикуется в теме /throttle_controller/filtered_velocity для облегчения наблюдения.

Эффекты фильтрации и контроля:



Красная линия на рисунке — это данные, полученные с одометра, а голубая линия — отфильтрованные данные. Видно, что после фильтрации по Гауссу сигнал скорости имеет небольшую ошибку на этапе постоянной скорости, но существует определенная ошибка на этапах ускорения и замедления. Это связано с тем, что введение фильтра вызывает определенную задержку по времени.