Отчет по лабораторной работе N6.

Группа:
1) Алхименков Леонид
2) Казаков Максимилиан
3) Хуан Бинкунь

СЛАМ, mapping, amcl

SLAM (одновременная локализация и картирование) — это вычислительная задача, которая включает в себя создание карты неизвестной среды с одновременным отслеживанием местоположения робота в этой среде. Эта проблема обычно встречается в робототехнике, где роботу необходимо ориентироваться в неизвестной среде при построении карты окружения.

Картирование в робототехнике относится к процессу создания представления окружающей среды с использованием данных датчиков. Это важный шаг в автономной робототехнике, поскольку он дает роботу понимание своего окружения. Процесс картирования включает в себя сбор данных с датчиков, таких как LiDAR, камеры или гидролокатор, и использование этих данных для построения представления об окружающей среде.

AMCL (адаптивная локализация Монте-Карло) — это алгоритм локализации, обычно используемый в робототехнике для отслеживания местоположения робота в окружающей среде. Он использует фильтр частиц для оценки положения робота на основе измерений датчика. AMCL особенно полезен в ситуациях, когда начальное положение робота неизвестно или когда положение робота необходимо отслеживать в течение длительного периода времени.

Алгоритмы отслеживание траектории

Алгоритмы отслеживания траектории используются в робототехнике и системах управления, чтобы заставить систему следовать по желаемой траектории. Вот некоторые распространенные алгоритмы отслеживания траектории:

1. Пропорционально-интегрально-дифференциальное (ПИД) управление. Это классический алгоритм управления, который регулирует выходной сигнал системы на основе ошибки между желаемой траекторией и фактическим выходным сигналом системы. Он широко используется в робототехнике и автоматизированном производстве.

2. Модель прогнозирующего управления (MPC): MPC — это алгоритм управления, который предсказывает будущее поведение системы и соответствующим образом

корректирует выходные данные системы. Он широко используется в автономных транспортных средствах и самолетах.

3. Управление режимом скольжения (SMC): SMC представляет собой алгоритм управления, который создает поверхность скольжения между желаемой траекторией и фактическим выходным сигналом системы. Выход системы затем корректируется для поддержания поверхности скольжения. Он широко используется в электродвигателях и силовых системах.

4. Адаптивное управление. Алгоритмы адаптивного управления регулируют выходные данные системы в зависимости от изменений в системе или окружающей среде. Они широко используются в аэрокосмической и оборонной промышленности.

5. Управление на основе нечеткой логики. Алгоритмы управления на основе нечеткой логики используют нечеткие наборы и логику для настройки выходных данных системы на основе ошибки между желаемой траекторией и фактическим выходным сигналом системы. Они широко используются в робототехнике и автоматизированном производстве.

Постановка задачи

1. Реализвать в patrol_bot логику обхода роботом нескольких точек заданных пользователем. Т.е. пользователь накликивает с помощью PublishPoint заданное колличество точек (или любое количество с возможностью пополнения), после чего робот обходит их последовательно, отправляя на исполнение каждую следующую точку только после достижения предыдущей. Обеспечить заданне угловое положение в точке, равным углу между текущим положением и заданной целью.
2. Зациклить движение (после достижения последней точки - продолжить с начала)

Предлагаемое решение

Мы записываем в массив точки и движемся по ним последовательно.

Код решения

```
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <geometry_msgs/PointStamped.h>
#include <actionlib/client/simple_action_client.h>
#include <vector>

using
MoveBaseClient=actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>;
//умный указатель на объект - клиент
boost::shared_ptr<MoveBaseClient> moveBaseClientPtr;
```

```
std::vector<geometry_msgs::PointStamped> route_vector = {};
int curr_point_ix = 0;
bool goal_achieved = true;

void active_callback()
{
   ROS_INFO_STREAM("goal is started");
}

void feedback_callback(const move_base_msgs::MoveBaseFeedbackConstPtr& feedback)
{
   ROS_INFO_STREAM("feedback "<<
                   " robot pose x" << feedback->base_position.pose.position.x <<
                   " y = "<<feedback->base_position.pose.position.y);
}

void done_callback(const actionlib::SimpleClientGoalState& state,
                   const move_base_msgs::MoveBaseResultConstPtr& result)
{
   if (state == actionlib::SimpleClientGoalState::SUCCEEDED)
   {
      ROS_INFO("Target is reached");
      curr_point_ix += 1;
      if(curr_point_ix > route_vector.size() - 1) {
         curr_point_ix = 0;
      }
      goal_achieved = true;
      ROS_DEBUG_STREAM("Current target" <<route_vector[curr_point_ix].point.x<<"
"<<route_vector[curr_point_ix].point.y);
   }

   else
   {
      ROS_ERROR("move_base has failed");
   }
}


void clickPointCallback(const geometry_msgs::PointStamped& point)
{
   ROS_INFO_STREAM(" get point "<<point.point.x<<" "<<point.point.y);
   route_vector.push_back(point);
}
```

```
void nextGoal(){
   if(goal_achieved && route_vector.size() > 0){
      goal_achieved = false;

      //задаем ориентацию в целевой точке
      double target_angle = M_PI/2;

      //формируем структуру цели для move_base Action
      move_base_msgs::MoveBaseGoal goal;
      goal.target_pose.pose.position = route_vector[curr_point_ix].point;
      //задаем кватернион, соответствующий ориентации
      goal.target_pose.pose.orientation.z = sin(target_angle/2);
      goal.target_pose.pose.orientation.w = cos(target_angle/2);
      goal.target_pose.header.frame_id = "map";
      goal.target_pose.header.stamp = ros::Time::now();
      //отправляем цель
      moveBaseClientPtr->sendGoal(goal,
                           done_callback,
                           active_callback,
                           feedback_callback
                           );
   }
}

int main(int argc, char **argv)
{

   ros::init(argc, argv, "control_node");


   ros::NodeHandle node("~");

   moveBaseClientPtr.reset(new MoveBaseClient("/move_base", false));

   while( !moveBaseClientPtr->isServerConnected())
   {
      ros::spinOnce();
   }

   ROS_INFO_STREAM("server move_base is connected");

   ros::Subscriber point_sub = node.subscribe("/clicked_point", 1, clickPointCallback);
```

```
    ros::Rate r(10);
    while(true) {
        nextGoal();
        ros::spinOnce();
        r.sleep();
    }

    return 0;
}
```

Результаты

Модель движется последовательно по заданным точкам в соответствии с заданием.
Ссылка на результаты: https://cloud.mail.ru/public/LxLR/kFrWoK6Gm

Литература

1. Курс лекций по «Программное обеспечение управляющих комплексов» 2023г.
2. Robot Operating System for Absolute Beginners(2018) Lentin Joseph.