РАЗРАБОТКА СИСТЕМЫ ЛОКАЛИЗАЦИИ ДЛЯ МОБИЛЬНОГО РОБОТА С ПРИМЕНЕНИЕМ ГРАФИЧЕСКОГО ПРОЦЕССОРА

М.Н. Рудь, А.Р. Пантюхин
Томский политехнический университет
rudmax13@gmail.com, sanyapantiukhin@gmail.com

Введение

Одной из важнейших задач, решаемых автономным мобильным роботом, действующим в заранее известной среде, является локализация (определение своего местоположения в пространстве). В данной работе используется алгоритм, основанный на фильтре частиц (метод Монте – Карло) [1].

Метод Монте – Карло основан на представлении состояния системы в виде набора частиц. Каждая частица – объект, который содержит информацию о вероятном местоположении и направлении движения робота, а также дополнительный параметр, именуемый *весом частицы*. Частицам предоставляется карта местности в виде сетки занятости (*occupancy grid*). Каждый цикл получения данных с сенсоров вес частицы вычисляется в зависимости от того, насколько точно показания «виртуальных» сенсоров частицы совпадают с реальными показаниями сенсоров робота. После этого происходит отбор наиболее сильных частиц. В результате, через определенное количество циклов движения и получения информации с сенсоров, в системе останутся частицы, показавшие наилучшие результаты. Они будут сконцентрированы вокруг реального положения робота и смогут достоверно его определить.

Чтобы получить необходимую точность локализации, необходимо охватить как можно больше вероятных состояний робота, что ведет к увеличению числа частиц в системе. Это делает применение фильтра частиц в реальном времени очень затратным с точки зрения скорости вычислений. Необходимо отметить, что большинство шагов фильтра выполняются независимо для каждой частицы, что позволяет осуществлять параллельную обработку всех частиц.

В последние годы графические процессоры (GPU) трансформировались из устройств для вывода компьютерной графики в мощные многоядерные аппаратные решения для осуществления параллельных вычислений общего назначения. При решении задач, где возможно осуществить параллельную обработку данных (к таким задачам относится, например, обработка изображений и фильтр частиц, рассматриваемый в данной статье) можно получить прирост производительности в десятки раз по сравнению с традиционной реализацией вычислений на центральном процессоре (CPU).

CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений на графических процессорах (GPU) компании NVIDIA [2].

В данной работе на архитектуру CUDA был перенесен наиболее ресурсоемкий шаг фильтра частиц – вычисление весов, который выполняется независимо для каждой частицы и пригоден для параллельного исполнения. Для повышения наглядности полученных результатов алгоритм был реализован как на многоядерном CPU, так и на устройстве с поддержкой CUDA – мобильной видеокарте GeForce GT 640M.

Аппаратная часть

Используемый робот имеет два ведущих колеса. В качестве основного сенсора используется Microsoft Kinect [3]. Kinect предоставляет видеоизображение, комбинированное с соответствующей ему картой глубины, содержащей расстояния до каждой точки обозреваемого пространства. Сенсор имеет разрешение 640 x 480 точек и угол обзора приблизительно 60 градусов.

Модель вычислений CUDA

Технология CUDA реализует модель вычислений типа «сетка». Низшими звеньями в этой модели выступают так называемые «нити», которые выполняют элементарные операции, например, сложение элементов двух векторов. Нити объединяются в блоки, причем мы можем пользоваться тремя измерениями. На практике обычно используются одномерный или двухмерный случаи. Блоки, в свою очередь, объединяются в сетку (*grid*), которая имеет два измерения.

Алгоритм

Алгоритм, используемый в работе, основан на использовании фильтра частиц. Каждая частица в системе представляет собой одно из возможных состояний мобильного робота. Состояние робота включает три переменных: координаты *x, y* центра масс робота, а также угол рысканья θ. Для каждого цикла движения и считывания информации с сенсоров состояния обновляются с учетом поданных на моторы команд, показаний электронного компаса и сенсора Kinect. Далее будут описаны основные шаги алгоритма Монте – Карло для осуществления локализации робота, а также даны комментарии по реализации алгоритма на GPU.

Инициализация начальных состояний частиц.

Изначально придаем переменным *x, y* и θ для всех частиц случайные значения.

Предсказание следующего положения робота.

На данном шаге выполняется чтение показаний энкодеров двигателей. Показания энкодеров приходят в виде количества градусов, на которые повернулся двигатель. Для того, чтобы предсказать следующее положение робота, необходимо из текущего показания энкодеров вычесть показания энкодеров на предыдущем цикле считывания. Это число градусов должно быть переведено в расстояние, пройденное роботом за это время, с учетом диаметра колес. Наконец, необходимо прибавить полученное число к предыдущей позиции робота. Учитывая наличие шумов в энкодерах, в модель движения должен быть также добавлен случайный шум.

Для реализации данного шага на GPU мы должны запустить ядро (специальное название для функции, выполняемой на GPU), с количеством нитей равным количеству частиц. Каждая нить ядра выполняет все необходимые вычисления для соответствующей ей частицы.

Вычисление весов частиц

Как было отмечено ранее, вес частицы зависит от того, насколько точно показания «виртуальных» датчиков частицы совпадают с реальными показаниями сенсора Kinect. Исходя из этого, необходимо создать модель сенсора Kinect, которой могли бы пользоваться частицы для получения своих показаний. Для создания модели сенсора карта глубины разбивается на 36 регионов. Для каждой ячейки размера 107 x 80 находится среднее значение глубины путем сложения значений всех пикселей и деления полученной суммы на общее число пикселей. Полученная матрица размера 6 x 6 разбивается на столбцы. Для каждого из 6 столбцов находится минимальное значение. Таким образом, мы получаем своеобразный набор из 6 дальномеров, лучи которых расположены по дуге с промежутком 10 градусов.

Перейдем непосредственно к вычислению весов частиц. В начале, инициализируем веса частиц равными единицам. Затем, будем последовательно умножать этот вес на следующее выражение:

$$1.0-(\frac{|Mr_i - Mp_i|}{Mr_i + Mp_i}),$$

где Mr_i - показание i-го датчика робота, Mp_i - показание i-го датчика частицы.

Частицы, наиболее близкие к реальному положению робота, будут иметь наибольший вес, наиболее удаленные частицы – наименьший.

Данный шаг реализуется на GPU аналогично предыдущему. Мы запускаем ядро, где каждая нить отвечает за вычисление веса для соответствующей частицы в наборе. На вход ядра поступает вектор данных с сенсора Kinect, а также карта помещения, размещенная в текстурной памяти GPU. После проведения вычислений, массив с весами частиц передается в оперативную память для проведения отбора, который на данной стадии работы выполняется полностью на CPU.

Отбор частиц

На этом шаге необходимо провести отбор наиболее сильных частиц. Общее число частиц при этом должно сохраняться неизменным, поэтому некоторые частицы попадут в новый набор несколько раз.

Определение местоположения робота

После получения нового набора частиц, вычислим предполагаемые координаты x, y, θ робота путем вычисления средних значений данных переменных между всеми частицами.

Результаты

В качестве тестового стенда был использован ноутбук с процессором Intel Core i5 3210M и мобильной видеокартой NVIDIA GeForce GT 640M. Для наглядности алгоритм был протестирован на нескольких значениях количества частиц в системе. Результаты, полученные при реализации алгоритма на CPU представлены на рис. 5; результаты, полученные с ускорением на GPU, представлены на рис. 1:

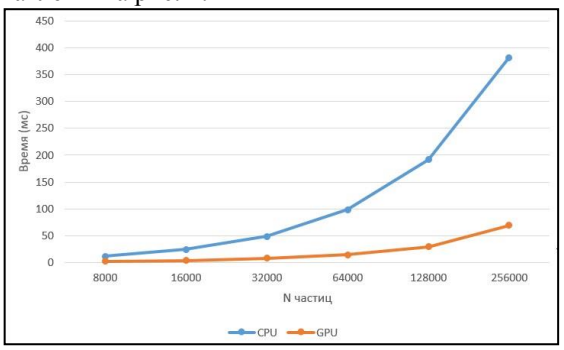

Рис.1. Графики производительности на CPU и GPU

Как можно увидеть из графиков, представленных выше, при ускорении алгоритма на GPU нами было получено повышение производительности более чем в 15 раз по сравнению с реализацией на CPU. Такой значительный прирост связан в первую очередь с переносом на GPU вычисления весов частиц, которое обладает высокой степенью параллелизма. В дальнейшем планируется протестировать алгоритм на одноплатном компьютере Jetson TK1, а также интегрировать систему локализации в единую систему управления автономным роботом.

Литература

1. S. Thrun «Probabilistic robotics» // Wolfram Burgard, 2000
2. N. Wilt «CUDA Handbook: A comprehensive guide to GPU programming» // Addison-Wesley, 2013
3. S. Falahati «OpenNI Cookbook» // PACKT Publishing, 2011