Архитектуры систем управления роботами: теория и практика

Ярмолинский Арсений Маркович

студент магистратуры СПбПУ им. Петра Великого
учитель информатики
педагог дополнительного образования

Три слона


РОБОТ
состоит из:
Механическая часть
Система питания
Электронная система управления
Механическая конструкция
Приводы
Управляющий контроллер
Система датчиков
Индикация
Программное обеспечение

Что такое архитектура системы управления роботом?

Термин “*архитектура системы управления*” используется для обозначения того, как система делится на подсистемы и как эти подсистемы взаимодействуют.

Архитектура СУ роботов отличается от других архитектур программного обеспечения особыми потребностями роботизированных систем.

Требования к роботам

Каковы особые требования к роботизированным системам?

Роботизированные системы работают в сложных динамичных средах реального времени. Эти системы должны:

- управлять различными датчиками и исполнительными механизмами в режиме реального времени,
- делать это в условиях значительной неопределенности и шума, отслеживая неожиданные ситуации и реагируя на них,
- и делать все это одновременно и асинхронно.

Кроме того, роботизированные системы должны реагировать в *различных временных рамках* - от миллисекундного управления с обратной связью до минут или часов, для решения сложных задач.

История исследований

Разработка архитектуры и программирования роботов началась в конце 1960-х годов с создания робота Shakey в Стэнфордском университете.

На рисунке: робот Shakey. У Shakey была камера, дальномер и датчики ударов, и он был подключен к компьютерам DEC PDP-1O и PDP-l5 по радио- и видеосвязи.

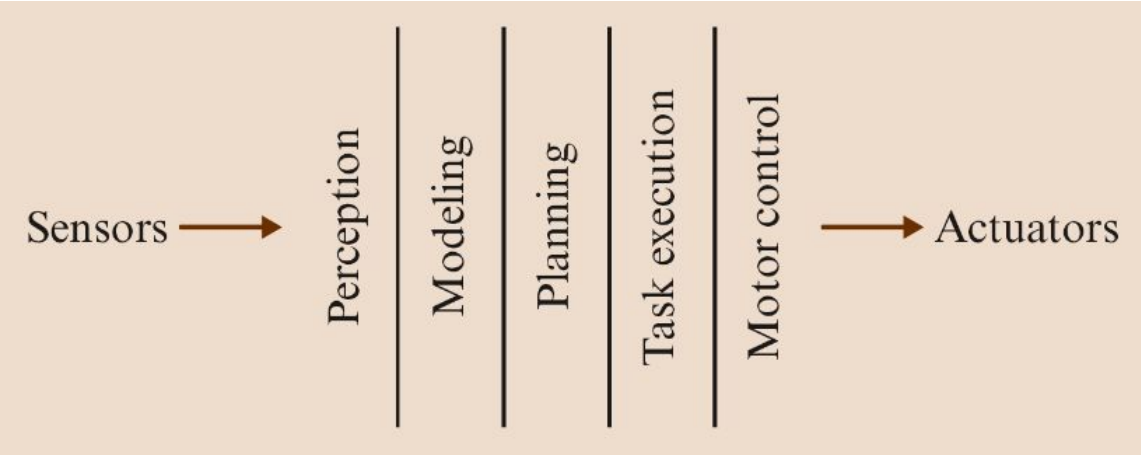
Sense-Plan-Act

Архитектура Shakey была разделена на три функциональных элемента: **Sense**, **Plan** и **Act**.

1. Система распознавания преобразовывает изображение с камеры во внутреннюю модель мира.
2. Планировщик берет построенную модель мира и текущую цель и генерирует план (т.е. последовательность действий), который позволит достичь цели.
3. Исполнитель берет план и посылает действия на исполнительные механизмы робота.

Sense-Plan-Act

Этот подход получил название парадигмы “sense-plan-act” (SPA).



Компоненты робота в этом случае, как говорят, организованы горизонтально. Информация из окружающего мира в виде данных сенсоров и датчиков должна пройти несколько промежуточных этапов интерпретации, прежде чем, наконец, стать доступной для выдачи управляющих воздействий.

Sense-Plan-Act

Основные архитектурные особенности SPA-парадигмы заключаются в том, что данные с датчиков используются для построения модели мира, которую впоследствии использует планировщик для формирования плана действий, исполняемого в дальнейшем исполнителем *не используя напрямую данные с датчиков, которые использовались для построения модели*.

Sense-Plan-Act

- В этих ранних системах основное внимание уделялось *созданию подробной модели мира*, а затем тщательному *планированию* дальнейших шагов.
- Проблема заключалась в том, что, пока робот создавал свою модель и размышлял о том, что делать дальше, мир, скорее всего, менялся.
- Таким образом, эти роботы демонстрировали странное поведение: они смотрели (собирали данные, часто в виде одного или нескольких снимков с камеры), обрабатывали и планировали, а затем (часто после значительной задержки) приступали к действию на пару шагов, прежде чем начать цикл сначала (*look and lurch behaviour*).

Проблемы с архитектурой SPA

1. В реальном мире создание плана занимает продолжительное время в течение которого робот заблокирован и не реагирует на возможные изменения мира.
2. Исполнение плана без датчиков обратной связи может быть опасно для успешности выполнения задания или для самого робота.

Может быть так, что система SPA разработает план, но прежде чем этот план может быть выполнен в полном объеме, он становится недействительным из-за изменений в реальном мире.

Реактивная архитектура

В 1986 году Родни А. Брукс опубликовал статью, в которой описывался тип реактивной архитектуры, называемый архитектурой подчинения (subsumption architecture).

Реализация

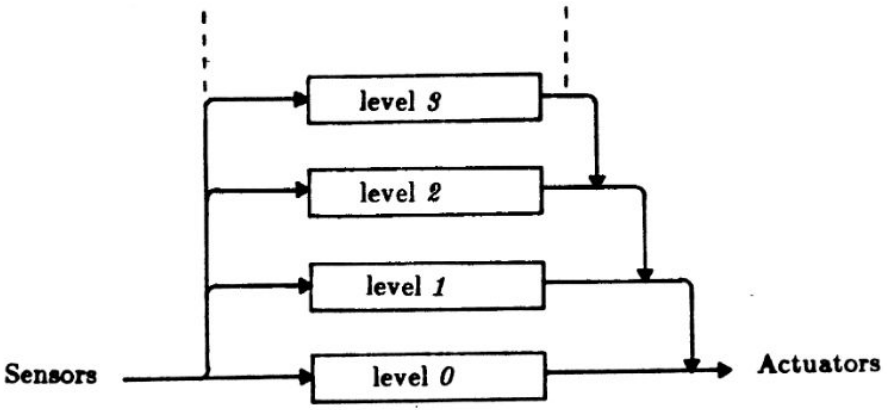
- Архитектура подсистемы построена на основе *уровней* взаимодействующих **конечных автоматов**, каждый из которых напрямую соединяет датчики с исполнительными механизмами.
- Эти конечные автоматы назывались *поведения* (*behaviors*) (что привело к тому, что некоторые назвали архитектуру subsumption - behavior-based или поведенческой робототехникой).
- Поскольку в любой момент времени могло быть задействовано несколько моделей поведения, у subsumption был механизм **арбитража**, который позволял моделям поведения **более высокого уровня переопределять сигналы с более низкого уровня поведения**.

Архитектура подчинения на время стала доминирующим подходом в архитектурах реактивных роботов.

Архитектура подчинения (subsumption)

Архитектура подчинения (Brooks, 1986) характеризовалась:

1. отсутствие представления о внешнем мире,
2. фокус на отдельных задачах, а не системе целиком,
3. подчинение низкоуровневых поведений поведениями более высокого уровня,
4. плотная связь между датчиками и исполнительными механизмами.



Пример

1. Например, поведение робота может заключаться в том, что он просто управляет роботом в произвольных направлениях. Это поведение всегда активно, и робот всегда куда-то направляется.
2. Поведение второго, более высокого уровня может принимать сигналы датчиков, обнаруживать препятствия и уводить робота от них. Он также всегда активен.

В среде, где нет препятствий, поведение более высокого уровня никогда не генерирует сигнал. Однако, если он обнаруживает препятствие, он переопределяет поведение более низкого уровня и уводит робота в сторону.

Как только препятствие исчезнет (и более высокий уровень поведение перестает посылать сигналы), поведение на более низком уровне снова получает контроль. Для создания все более сложных роботов можно создать множество взаимодействующих уровней поведения.

Сравнение архитектуры подчинения и SPA

- *Масштабируемость:* Брукс утверждал, что там, где архитектура SPA должна быть существенно переработана, чтобы обеспечить включение новых возможностей в роботизированную систему, архитектура subsumption позволяла добавлять новые возможности, просто добавляя в систему новые уровни поведения, которые могли переопределять или включать в себя поведение более низких уровней там, где это необходимо, без необходимости вмешиваться или перепроектировать поведение более низких уровней.

- *Производительность:* В то время как SPA-роботы были медленными и громоздкими, роботы с использованием subsumption были быстрыми и реактивными. Динамичный мир их не беспокоил, потому что они постоянно взаимодействовали с датчиками, чувствовали окружающий мир и реагировали на него.

Проблемы с реактивным (поведенческим) подходом

Однако роботы, основанные на поведении, вскоре достигли предела своих возможностей.

1. С таким подходом оказалось очень трудно получить правильную комбинацию поведений для достижения **долгосрочных целей**,
2. Оказалось, что оптимизировать поведение робота практически невозможно (*здесь имеется ввиду математический смысл слова оптимальный*).

Гибридная архитектура

По сути, роботам требовались возможности ранних архитектур строить долгосрочные планы, также как и реактивное поведение behaviour-based систем.

Осознание этого привело к разработке ***многоуровневых*** архитектур управления роботами.

Гибридная архитектура

Эти гибридные архитектуры могут быть охарактеризованы распределением задач по уровням, где уровни низкого уровня обеспечивают реактивное поведение, а уровни высокого уровня обеспечивают более интенсивные с точки зрения вычислений возможности долговременного планирования.

Наиболее популярным вариантом этих гибридных архитектур являются **трехуровневые архитектуры**:

1. Контроллер или Реактивный уровень
2. Секвенсор или Исполнительный уровень
3. Планировщик или Совещательный уровень

Контроллер (реактивный уровень)

- Уровень контроллера (он же реактивный) обеспечивает низкоуровневое управление роботом. Для него характерна тесная взаимосвязь между датчиками.
- Цикл принятия решения часто составляет порядка **миллисекунд**.
- С точки зрения разработки программного обеспечения контроллер представлял бы собой набор драйверов с базовыми реакциями, в то время как с биологической точки зрения контроллер - это набор нервных связей с мышцами и другими органами.
- Управляющие элементы должны обладать низкой вычислительной сложностью, чтобы они могли быстро реагировать на раздражители и быстро выполнять основные действия.

Секвенсор (исполнительный уровень)

- Уровень секвенсора (он же исполнительный) находится между уровнем контроллера низкого уровня и уровнем планировщика более высокого уровня.
- Он принимает директивы от уровня планировщика и упорядочивает их для реактивного уровня.
- Естественно, это упорядочение не может быть простой линейной программой, поскольку среда, в которой выполняется управление, может неожиданно измениться, и примитивное поведение контроллера может привести к сбою.
- Например, исполнительный уровень может обрабатывать набор промежуточных точек, сгенерированных планировщиком пути, и принимать решения о том, какое реактивное поведение следует активировать для движения в следующую точку.

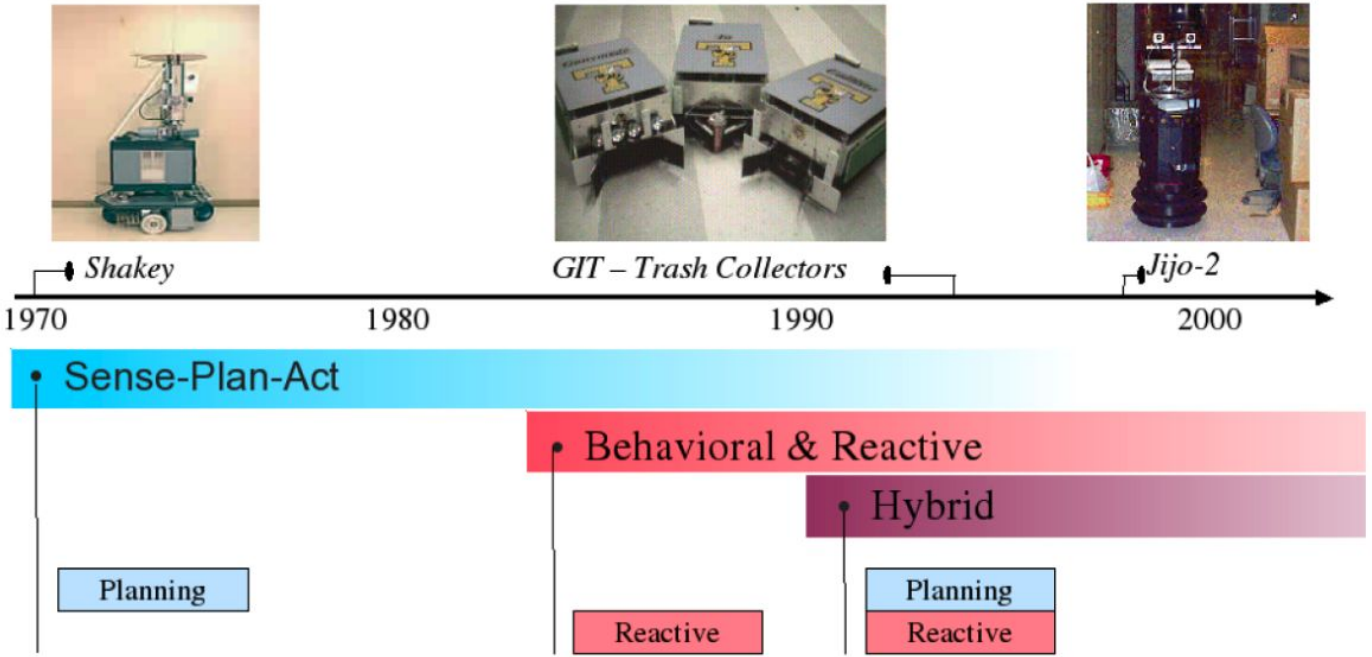
Секвенсор (исполнительный уровень)

- Уровень секвенсора также отвечает за интеграцию информации с датчиков во внутреннее представление состояния робота.
- Например, на нем могут размещаться процедуры локализации и картографирования.
- Циклы принятия решений на исполнительном уровне обычно занимают порядка **секунды**.

Планировщик (совещательный уровень)

- Планировщик, или совещательный уровень, содержит наиболее сложные вычислительные компоненты, традиционно содержащие затратные методы поиска в пространстве состояний с экспоненциальной или высокополиномиальной вычислительной сложностью.
- Планировщик генерирует глобальные решения для сложных задач.
- Цикл принятия решений в нем часто составляет **несколько минут**.
- Для принятия решений планировщик использует модели.
- Эти модели обычно используют информацию о состоянии, собранную на исполнительном уровне.

Временная шкала развития архитектур управления роботами



Пример кода SPA

```
1  task main() {
2    while(true) {
3      if(SensorValue(touchSensor)==0) {
4        motor[motorC]=100;
5        motor[motorB]=100;
6      } else {
7        motor[motorC]=100;
8        motor[motorB]=-100;
9        wait1Msec(1500);
10     }
11   }
12 }
```

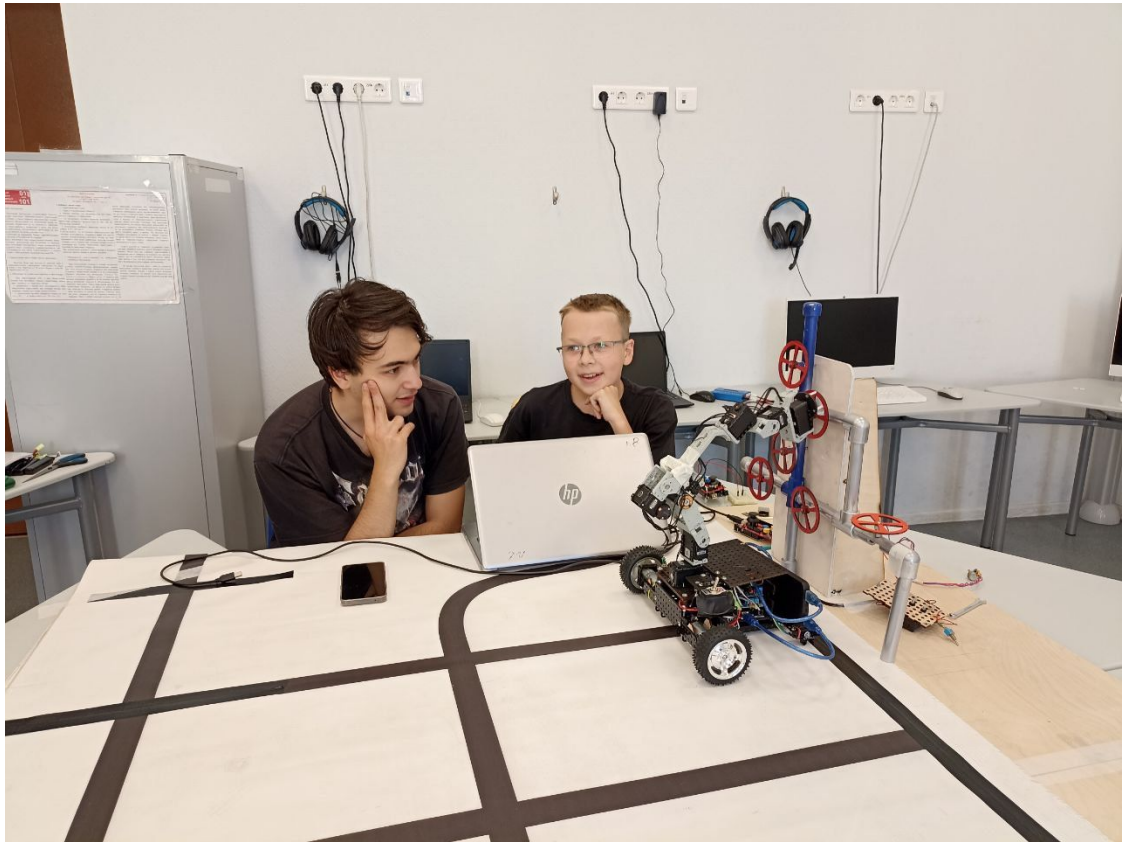
Пример кода SPA

```
void loop()
{
    ////// SENSE //////

    // Чтение сигнала с датчика расстояния
    float dist_forw = LIDAR.read();

    // Построение математической модели окружения
    bool is_free = dist_forw  > 20;


    ////// PLAN //////

    // Построение плана действий на основании
    //  математической модели
    if(is_free)
    {
        // Если впереди свободно - едем вперед
        m_left = 100;
        m_right = 100;
    }
    else
    {
        // Если впереди препятствие - отворачиваем
        m_left = -100;
        m_right = 100;
    }


    ////// ACT //////

    // Исполнение плана
    motors.run(m_left, m_right);
    delay(500);
}
```

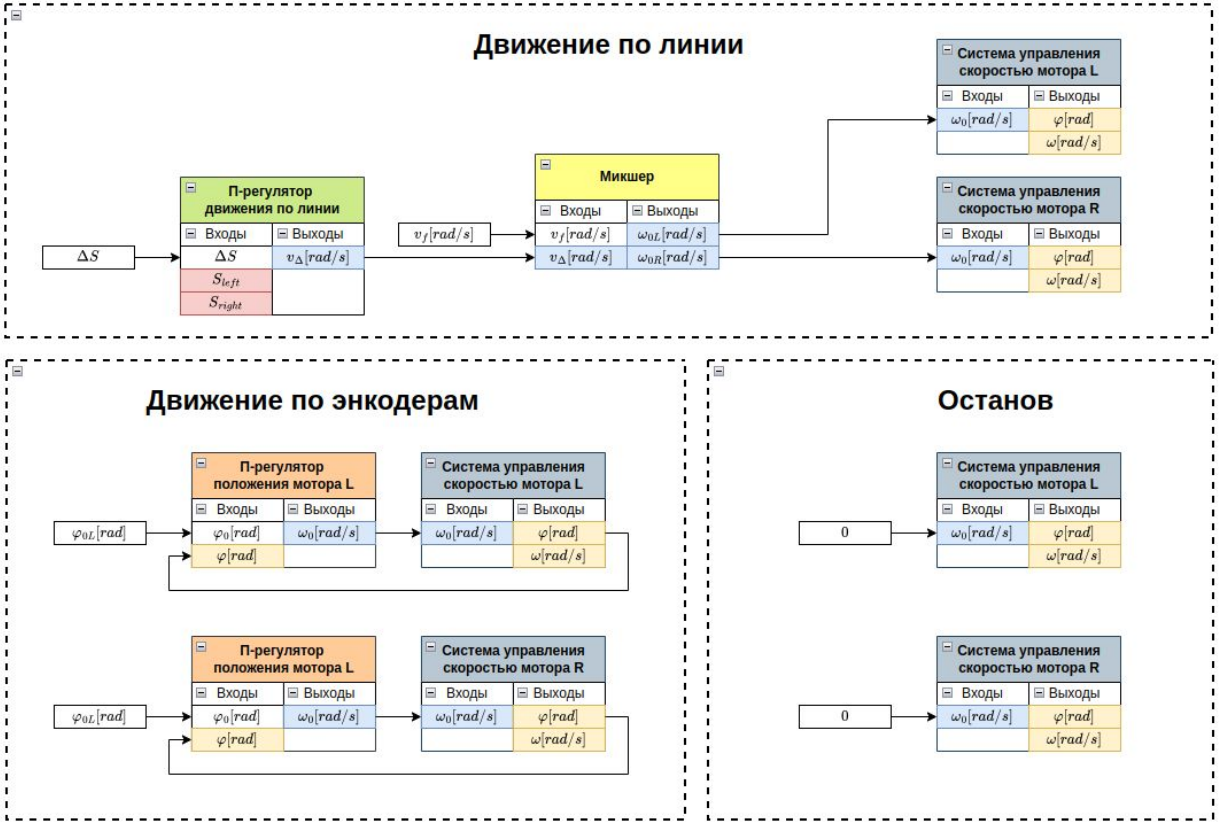
Практический пример: робот GLaDOS

Двухуровневая архитектура:

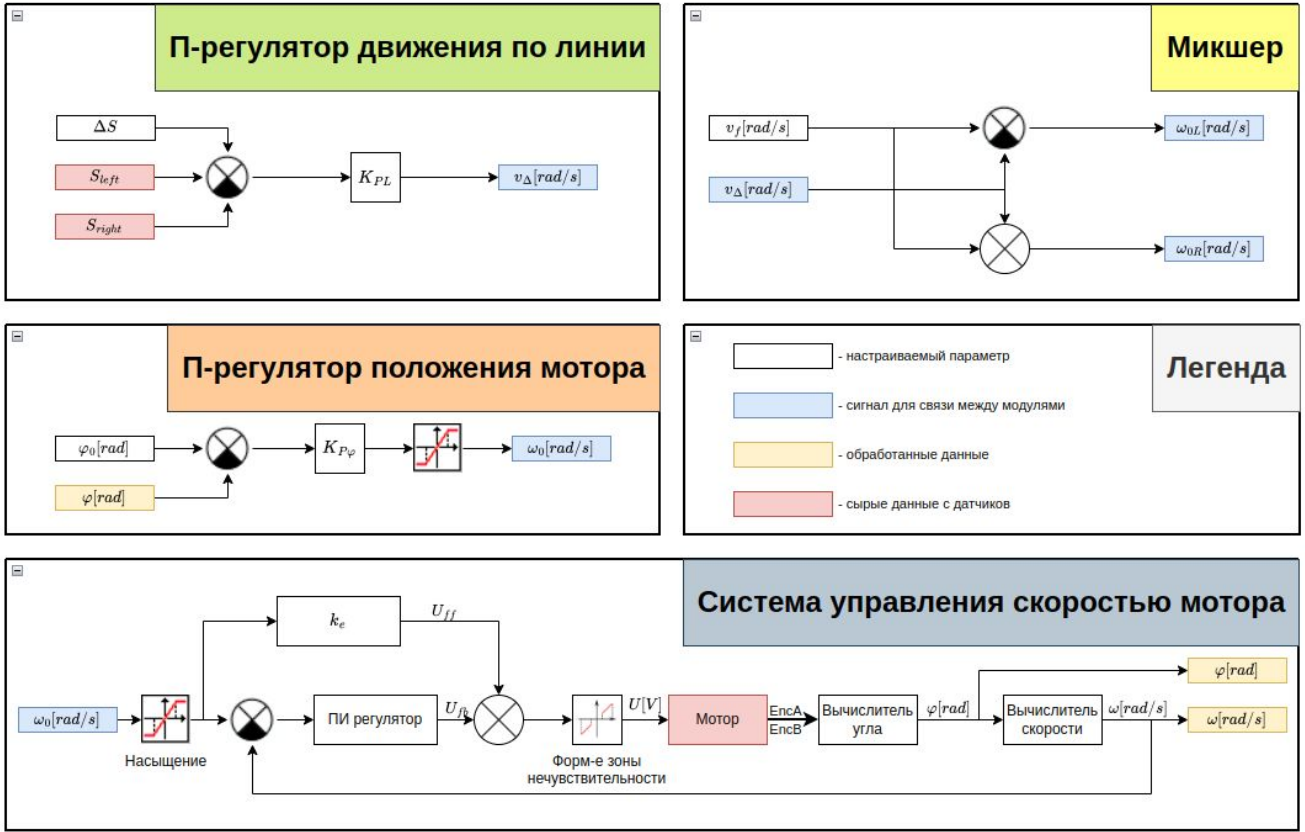
1. Секвенсор: идеология SPA
2. Контроллер: набор поведений



Контроллер: набор поведений



Контроллер: реализация передаточных функций

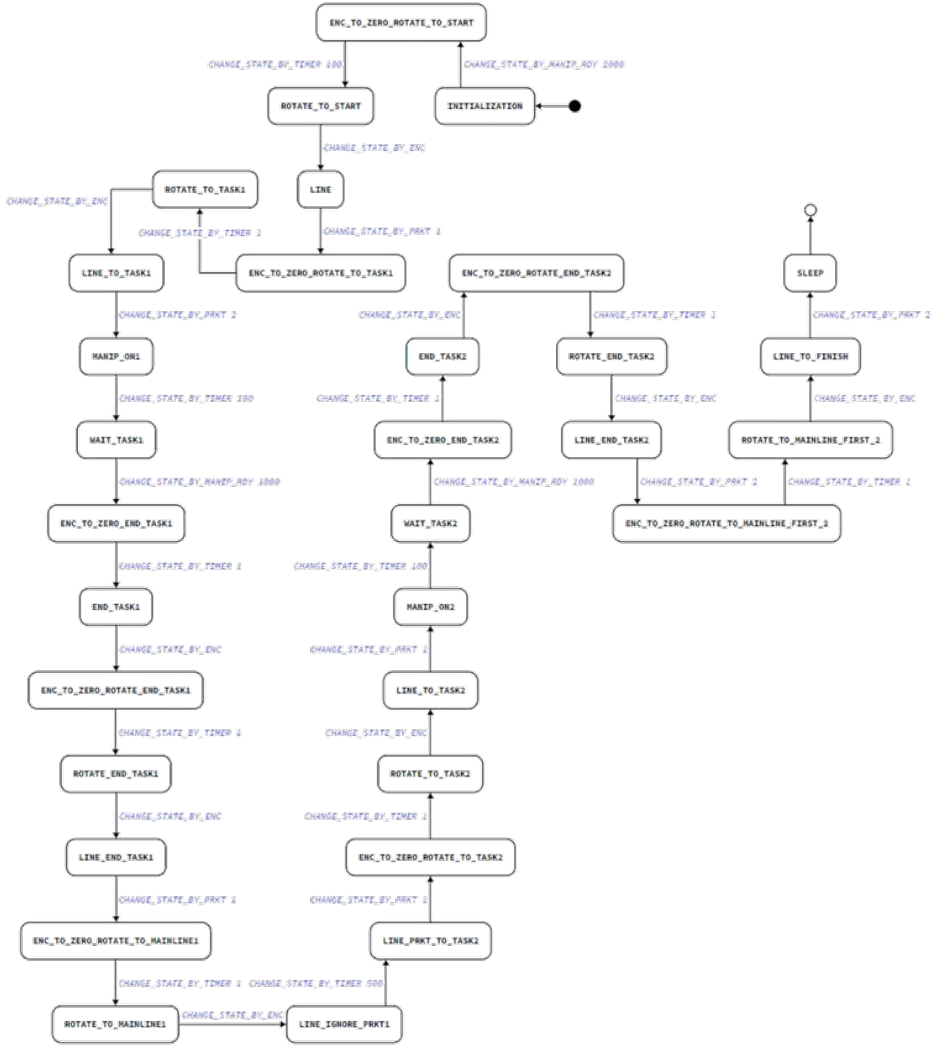


Секвенсор: SENSE

```
void loop()
{
  //////// ТАЙМЕР ////////
  while (micros() - tmr1 < Ts_us);
  dt = micros() - tmr1;
  tmr1 = micros();

  //////// SENSE ////////
  S_L = analogRead(S_LEFT);
  S_R = analogRead(S_RIGHT);
  s_per_l = digitalRead(S_PER_LEFT);
  s_per_r = digitalRead(S_PER_RIGHT);
  phi_l = motor_l.getAngle();
  phi_r = motor_r.getAngle();
  i_manip_rdy = digitalRead(I_MANIP_READY);
```

Секвенсор: PLAN

```
//////// PLAN ////////

v_L = 0;
v_R = 0;

switch (state)
{
case initialization:
  zero_intersections();
  change_state_by_manip_rdy(2000, enc_to_zero_rotate_to_start);
  break;
case enc_to_zero_rotate_to_start:
  zero_enc();
  change_state_by_timer(100, rotate_to_start);
  break;
case rotate_to_start:
  drive_deg(TURN_180, -TURN_180);
  change_state_by_enc(line);
  break;
case line:
  drive_line(v_f);
  // поверка перекрёстков
  count_intersections();
  // проверка перекрёстков (условие перехода)
  change_state_by_prkt(1, enc_to_zero_rotate_to_task1);
  break;
case enc_to_zero_rotate_to_task1:
```



Секвенсор: ACT

```
    break;
case sleep:
    break;
}


//////// ACT /////////
drive(v_L, v_R);
```

Контактная информация

telegram: @arsenis

email: yarmolinskiyam@gmail.com

Блог: https://arsenier.github.io/