Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

О Т Ч Ё Т П О Н А У Ч Н О - И С С Л Е Д О В А Т Е Л Ь С К О Й Р А Б О Т Е

Разработка алгоритма предотвращения столкновений манипулятора мобильного робота с препятствиями в режиме ручного управления

Выполнил
студент гр.3331506/60401 <*подпись*> Н.А. Халявин

Руководитель
доцент, к.т.н. <*подпись*> В.Н. Уланов

Научный консультант <*подпись*> Р.Р. Хазанский

«___» _____ 2019 г.

Санкт-Петербург
2019

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт металлургии, машиностроения и транспорта

Высшая школа автоматизации и робототехники

ЗАДАНИЕ

на выполнение научно-исследовательской работы

_____

студенту гр. _____

1 Тема работы

_____

_____

2 Срок сдачи студентом законченной работы

_____

3 Исходные данные к работе

_____

4 Содержание расчетно-пояснительной записки

_____

5 Перечень графического материала (с точным указанием обязательных чертежей)

_____

6 Консультанты по работе

_____

7 Дата выдачи задания _____

Руководитель _____

(ФИО, должность, подпись руководителя)

Задание принял к исполнению

_____

(дата)

_____

(подпись студента)

Оглавление

1. Введение

Современные робототехнические средства часто используются для выполнения работ в местах, куда по каким-либо причинам нельзя отправить людей. Успех выполнения задачи зачастую зависит от состояния манипулятора, но в случае его поломки бывает невозможно отправить на место людей для ремонта. В то же время управление манипулятором в большинстве случаев осуществляется оператором. По ряду причин, таких как отсутствие прямой видимости, задержки в линиях связи, а также просто ошибки оператора, велика вероятность столкновения звеньев манипулятора друг с другом и с корпусом робота, что может привести к поломке манипулятора. С целью предотвращения возникновения таких ситуаций в систему управления манипулятором необходимо встраивать средства предотвращения столкновений звеньев манипулятора с препятствиями.

На данный момент существуют различные подходы к решению этой задачи. Проблема их применения в мобильной робототехнике заключается в том, что обычно их программные реализации требуют большой производительности, а установка таких вычислительных систем на мобильных роботах осложнена из-за больших габаритов требуемых систем охлаждения. Выполнение же вычислений на удалённых вычислительных устройствах не позволяет гарантировать отсутствие столкновений, вызванных задержками в линиях связи.

По этой причине в данной работе производится разработка библиотеки функций на языке C++, использование которой позволяет решить проблему столкновений звеньев с использованием микроконтроллера stm32f407.

При этом к решению предъявляются следующие требования:

- решение не должно быть причиной столкновений, которых бы не произошло без его использования;
- решение должно обладать сравнительной универсальностью;
- точность позиционирования манипулятора не должна снизиться.

2. Аналитический обзор существующих подходов к решению проблемы

2.1. Аппаратные решения

Аппаратные решения подразумевают установку на звеньях манипулятора дополнительных датчиков, которые должны обнаруживать близость звеньев к препятствиям и корректировать параметры движения [1, 2]. Примером используемых типов сенсорных систем являются системы силомоментного очувствления, визуальные системы, а также датчики сближения (например, ультразвуковые, инфракрасные или ёмкостные) [1].

Такие системы имеют множество преимуществ [1, 2]:

- алгоритмы их работы просты, поэтому от вычислительного устройства не требуется большая производительность;
- не требуют внесения информации о геометрических характеристиках и расположении звеньев и препятствий;
- могут работать с недетерминированными объектами.

Однако установка дополнительных сенсорных систем на манипуляторе мобильного робота может быть затруднительной вследствие ограничений на габариты роботов. По этой причине такие системы в этой работе далее не рассматриваются.

2.2. Программные методы. Стратегии поведения

Программные решения подразумевают наличие алгоритма, который на основании информации о геометрических размерах звеньев и их расположении прогнозируют столкновения с препятствиями и корректируют скорость и/или направление их движения с целью предотвращения столкновений.

2.2.1. Потенциальное планирование

При использовании этого подхода манипулятор осуществляет движение в поле сил. Для предотвращения столкновений звеньев манипулятора с использованием стратегии потенциального планирования применяется метод отталкивающего потенциала. При этом потенциальное поле рассчитывается

таким образом, чтобы при сближении с препятствием возникала большая отталкивающая сила, но при нахождении звена далеко от препятствия влияние отсутствовало [3-6]. Пример функции зависимости потенциала от расстояния приведён в формуле 2.2.1.1 [4]:

$$U = \begin{cases} \frac{1}{2}\delta\left(\frac{1}{d} - \frac{1}{d_0}\right), d \le d_0 \\ 0, d > d_0 \end{cases}, \qquad 2.2.1.1$$

где d – расстояние между точкой пространства и препятствием, d_0 – граничное расстояние между звеном и препятствием, δ – коэффициент, позволяющий регулировать ускорение звеньев.

Фактически построение потенциального поля не выполняется, так как эта операция занимает много времени и требует больших объёмов памяти. Вместо этого выполняется вычисление виртуальных сил, действующих между каждой парой объектов, в зависимости от расстояния между ними. Зависимость виртуальной силы от расстояния приведена в формуле 2.2.1.2 [4]:

$$F = \begin{cases} \delta\left(\frac{1}{d} - \frac{1}{d_0}\right)\frac{1}{d^2}, d \le d_0 \\ 0, d > d_0 \end{cases}, \qquad 2.2.1.2$$

Таким образом, для решения задачи методом потенциального планирования необходимо вычислять расстояние между трёхмерными объектами. Для упрощения вычисления расстояния между звеньями они заменяются более простыми объектами, для которых удобно вычислять расстояние, например сферами [4], капсулами [4], или эллипсоидами [3].

Кроме отталкивающих сил используются также притягивающие силы, которые могут притягивать манипулятор к заранее определённой траектории либо к целевой точке. Это позволяет использовать такой подход не только для предотвращения столкновений, но и для планирования траектории движения манипулятора [3-6].

Главными достоинствами такого метода можно считать его простоту, а также плавность работы (так как в процессе работы симулируется действие сил, скорость не изменяется скачком). Также важным преимуществом является

возможность использовать метод для планирования траектории движения в автоматическом режиме.

Основным недостатком подхода является тот факт, что при его работе манипулятор может двигаться по траектории, отличной от заданной оператором. Поскольку метод работает только с препятствиями, для которых определены форма и местоположение, в результате такого отклонения траектории может произойти столкновение с препятствием, неизвестным системе, но известным оператору, что противоречит предъявленным к разрабатываемому решению требованиям. Также существенным недостатком системы является факт наличия ненулевого воздействия вблизи препятствий даже при нулевой скорости, что может привести к перемещению манипулятора после того, как оператор отпустит ручку управления.

2.2.2. Планирование в пространстве конфигураций

Вычисление расстояния между звеньями и прогнозирование столкновений в рабочем пространстве являются трудоёмкой задачей, поэтому данную задачу часто решают в пространстве конфигураций. В этом пространстве в качестве основных базисов выбраны вектора обобщённых координат манипулятора, причём конечная область пространства конфигураций описывает все возможные состояния [7].

Для решения поставленной задачи для каждой точки пространства конфигураций определяется наличие пересечения звеньев с препятствиями или друг с другом. Пример перехода к пространству конфигураций для плоского двухзвенного робота представлен на рисунке 2.2.2.1.

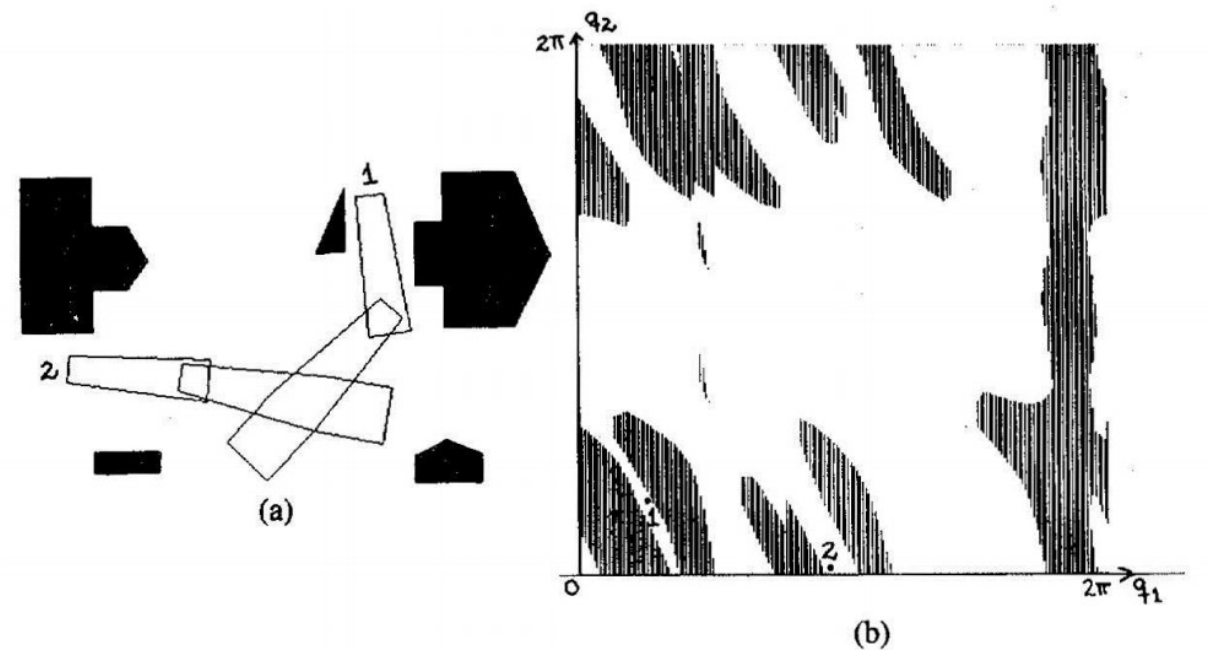

Рисунок 2.2.2.1 – Переход от рабочего пространства (a) к пространству конфигураций (b). Фото с сайта https://www.cs.columbia.edu/~allen/F19/NOTES/cspaceexamples.pdf

В дальнейшем пространство конфигураций можно разделить на дискретные области, а после на основе полученных данных построить граф конфигураций, для которого будет разрешён переход в вершины, соответствующие соседним областям пространства конфигураций, кроме тех в которых обнаружено пересечение [7]. В ходе работы алгоритма достаточно проверять, разрешён ли переход в вершину графа, в сторону которой направлен вектор желаемой скорости.

Достоинства данного подхода – быстрота и надёжность работы, а также возможность использовать тот же граф конфигураций для алгоритма автоматического планирования траектории.

Недостатком метода является необходимость поиска компромисса между дискретностью графа конфигураций и занимаемой памятью. Даже для трёхзвенного манипулятора при выборе всего 100 точек для каждой из его координат оказывается необходимо выделить $100 \cdot 100 \cdot 100 = 1000000$ бит памяти, то есть 125 килобайт, что превосходит объём ПЗУ большинства микроконтроллеров. Уменьшение же числа точек пространства приводит не

только к прерывистости движений манипулятора, но и к недостижимости многих промежуточных конфигураций манипулятора, что приводит к снижению точности работы манипулятора и увеличению энергетических затрат.

Также такой подход не позволяет учитывать столкновения с объектами, форма и расположение которых становятся известны только во время работы (например объекты, находящиеся в захватном устройстве).

2.3. Алгоритмы обнаружения пересечений трёхмерных объектов

Наиболее трудоёмкой задачей программных методов решения задачи является задача обнаружения столкновений (в англоязычной литературе – collision detection). Эта задача включает в себя обнаружение существующих и прогнозирование будущих столкновений, а также вычисление расстояния между объектами. Эта задача встречается в целом ряде применений, таких как тренажёры, симуляторы, физические движки для игр, а потому существует несколько подходов к её решению [8-16].

2.3.1. Априорный и апостериорный подходы

Используемые алгоритмы определения столкновений делятся на две группы: априорные и апостериорные [8, 10].

Априорные алгоритмы также называют непрерывными. Они прогнозируют траекторию движения объектов на основе информации о текущей скорости и действующих усилиях и таким образом прогнозируют время предстоящих столкновений с высокой точностью. При этом в момент столкновения происходит изменение траектории, что гарантирует отсутствие пересечений тел в любой момент времени. Такие алгоритмы характеризуются высокой точностью и надёжностью симуляций, однако построение траектории и точное определение времени пересечений требуют больших вычислительных затрат [8, 10].

Апостериорные же алгоритмы анализируют сцену на наличие уже произошедших столкновений объектов через дискретные промежутки времени. Такой подход требует меньших вычислительных затрат для обнаружения

столкновений, но с его помощью сложно прогнозировать будущие столкновения, так как необходимо производить расчёт множества последующих кадров, что требует больших вычислительных затрат и затрат памяти [8, 10]. Другой проблемой апостериорного подхода является необходимость разрешения существующих пересечений, которые возникают вследствие дискретности алгоритмов. Кроме того, такие алгоритмы не могут гарантировать отсутствие столкновений тел между кадрами (эта проблема носит название туннелирование). Для её предотвращения алгоритм необходимо модифицировать, добавляя проверку возможности столкновений между кадрами.

Поскольку целью работы является система предотвращения столкновений, точность моделирования поведения объектов во время столкновения не имеет решающего значения. Учитывая, что разрабатываемый алгоритм должен работать в условиях малой вычислительной мощности, применение априорных алгоритмов нецелесообразно, а потому в данной работе они не рассматриваются.

2.3.2. Геометрическая аппроксимация

Решение задачи обнаружения и прогнозирования пересечений трёхмерных объектов требует проверки каждой пары поверхностей (за исключением поверхностей, принадлежащих одному телу) и имеет квадратичную сложность выполнения. Поскольку для создания цифровых моделей трёхмерных тел зачастую используется аппроксимация большим количеством треугольных полигонов, для обнаружения коллизий требуются большие вычислительные мощности.

С целью уменьшения количества поверхностей вместо сложной модели объекта используется его аппроксимация ограничивающим объёмом – областью пространства, которая содержит в себе рассматриваемый объект целиком [8-12]. Обычно используются сферы, выровненные по осям прямоугольные параллелепипеды (в англоязычной литературе AABB), ориентированные прямоугольные параллелепипеды (OBB) и дискретно

ориентированные многогранники. Чаще всего в качестве ограничивающих объёмов используют выпуклые объекты, так как большинство алгоритмов обнаружения коллизий не может корректно обрабатывать невыпуклые объекты без их дополнительного разделения. Сложные объекты для повышения точности можно аппроксимировать несколькими простыми объектами. Поскольку построение геометрической аппроксимации ограничивающим объёмом является задачей с высокой вычислительной сложностью, оно обычно осуществляется до начала работы программы и за время её выполнения аппроксимация не изменяется.

2.3.3. Иерархические модели

С целью сокращения вычислительной сложности алгоритма он разделяется на отдельные иерархически зависимые фазы, каждая следующая фаза использует более детальную геометрическую аппроксимацию [10-13]. При этом вычислительная сложность на каждой следующей фазе существенно выше, чем на предыдущей, но переход к ней осуществляется только в том случае, если на более высоком уровне иерархии было обнаружено пересечение. Такой подход позволяет существенно снизить вычислительные затраты, не производя подробную проверку на пересечение находящихся далеко объектов. Пример иерархической аппроксимации представлен на рисунке 2.3.3.1.

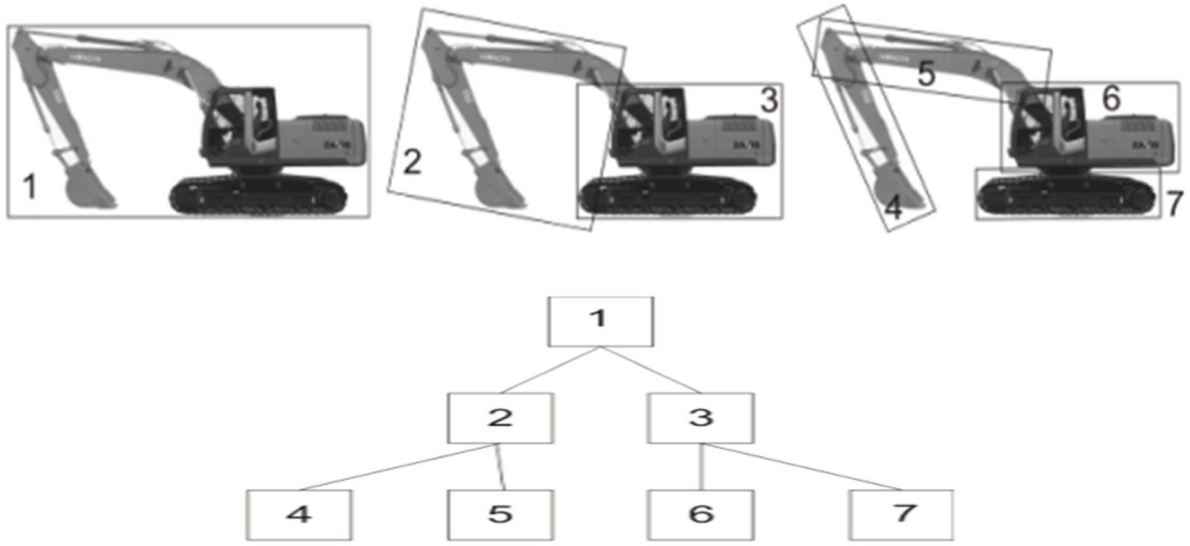

Рисунок 2.3.3.1 – Пример иерархической аппроксимации и дерево ограничивающих объёмов [10]

2.3.4. Алгоритмы обнаружения столкновений

Алгоритмы обнаружения коллизий во многом определяются выбранным типом ограничивающего объёма, рассматриваемого на конкретном шаге алгоритма. Так, для обнаружения коллизии двух сфер достаточно вычислить расстояние между их центрами и сравнить с суммой радиусов [8], для ориентированных по осям прямоугольных параллелепипедов – сравнить координаты их углов. Но также существуют алгоритмы, способные обнаружить коллизию любых выпуклых объектов.

Современные подходы во многом ориентированы на современное аппаратное обеспечение. Большое внимание уделяется алгоритмам, работающим на графических, а также многоядерных и многопоточных процессорах [14].

В данной работе будут рассмотрены два наиболее часто используемых алгоритма – алгоритм на основе теоремы о разделяющей оси (далее SAT) и алгоритм Гилберта-Джонсона-Кирти.

Алгоритм на основе SAT позволяет определить факт наличия или отсутствия коллизии двух объектов в конкретный момент времени. При этом теорема о разделяющей оси может быть сформулирована следующим образом: если существует такая прямая (для трёхмерного случая плоскость), что рассматриваемые объекты полностью располагаются по разные стороны от неё, то в этом и только в этом случае объекты не пересекаются [16]. Иллюстрация для двухмерного случая представлена на рисунке 2.3.4.1.

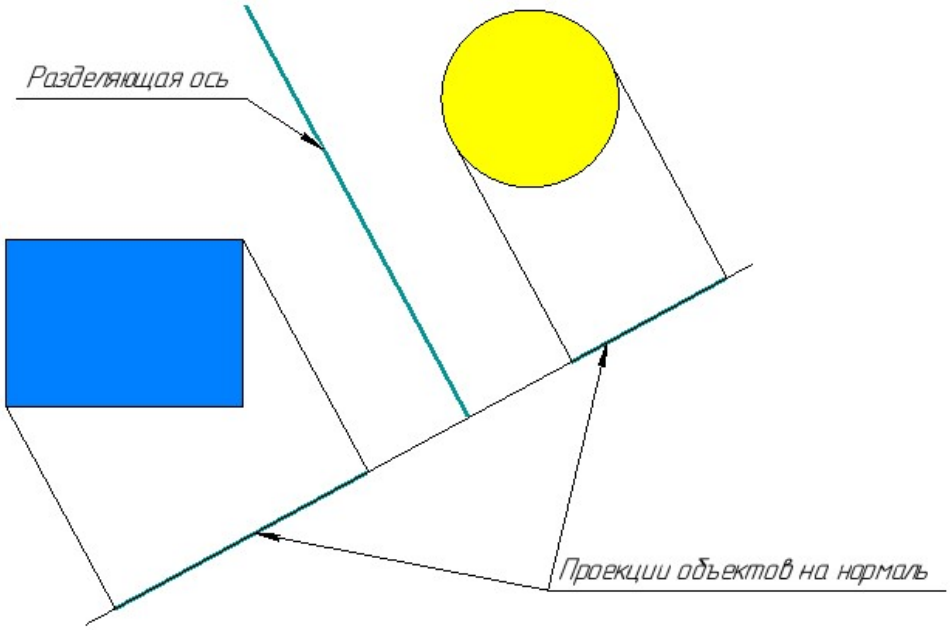

Рисунок 2.3.4.1 – SAT для двухмерного случая

Поскольку ортогональные проекции объектов на нормаль к этой плоскости также будут лежать по разные стороны от проекции самой плоскости, на практике проще искать не разделяющую плоскость, а нормаль к ней. То есть если удастся найти хотя бы один вектор такой, что проекции объектов на него не пересекаются, значит тела не пересекаются. При этом для объектов, включающих в себя только плоские поверхности, достаточно проверить конечное количество направлений.

Алгоритм Гилберта-Джонсона-Кирти представляет из себя итеративный алгоритм, позволяющий найти расстояние между двумя телами (или глубину проникновения, если тела пересекаются) [13, 15]. При этом он использует понятие разности Минковского [15]. Сумма Минковского двух множеств векторов – это множество, включающее в себя сумму каждого вектора одного множества с каждым вектором другого [9, 12, 15]. Определение суммы Минковского представлено в формуле 2.3.4.1.

$$A \oplus B = \{a + b,\ a \in A,\ b \in B\}, \qquad 2.3.4.1$$

Пример суммы Минковского для двумерных объектов представлен на рисунке 2.3.4.2.

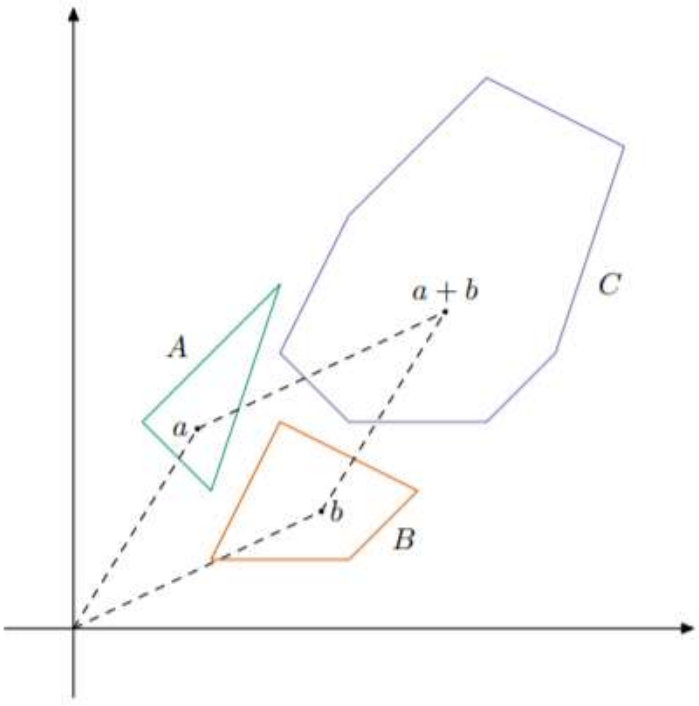

Рисунок 2.3.4.2 – Пример суммы Минковского для двумерных объектов

Фото с сайта https://codeforces.com/problemset/problem/1195/F?locale=ru

Разность Минковского представляет из себя сумму Минковского для первого множества и взятого с противоположным знаком второго множества.

Определение разности Минковского представлено в формуле 2.3.4.2.

$$A \ominus B = A \oplus (-B) = \{a - b,\ a \in A,\ b \in B\}, \qquad 2.3.4.2$$

Таким образом, если оба исходных множества содержали одну и ту же точку, разность Минковского для этих множеств содержит начало координат. Кратчайший вектор между началом координат и разностью Минковского объектов соответствует кратчайшему вектору между этими объектами.

В ходе работы алгоритма разность Минковского не вычисляется полностью, а ищутся только отдельные точки, необходимые на текущем шаге алгоритма. Они ищутся по определению разности Минковского, то есть как разность двух экстремальных точек исходных объектов в заданном направлении [15].

Для определения экстремальных точек используются опорные функции, то есть функции, возвращающие наиболее удалённую в заданном направлении

точку объекта [15]. Пример работы опорной функции представлен на рисунке 2.3.4.3.

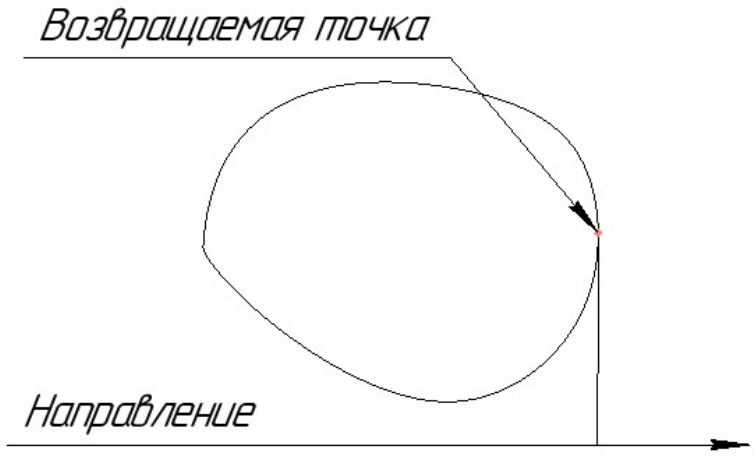

Рисунок 2.3.4.3 – Пример работы опорной функции

Опорные функции зависят только от геометрии объекта. Кроме того, ничто кроме них не зависит от геометрии объекта напрямую, что позволяет использовать одну и ту же реализацию алгоритма GJK для любых пар выпуклых объектов.

В ходе работы алгоритма GJK методом последовательных приближений ищется ближайшая к началу координат точка разности Минковского двух объектов. При этом используется понятие симплекса – простого объекта, содержащего от одной до четырёх (до трёх для плоского случая) точек. На каждом шаге происходит поиск точки симплекса, ближайшей к началу координат, и определяется направление от неё до начала координат. Затем в найденном направлении ищется новая экстремальная точка, и если её проекция на направление существенно больше проекции ближайшей к началу координат точки симплекса, то работа алгоритма завершается и возвращается расстояние до ближайшей точки симплекса. В противном случае новая точка добавляется в симплекс. Если симплекс переполнен, из него также удаляется старая точка, имеющая наименьшую проекцию на найденное направление [15].

3. Выбранный способ решения проблемы

3.1. Выбор стратегии решения

Из-за необходимости установки дополнительных сенсорных систем, которые могут быть слишком громоздкими для манипуляторов мобильных роботов, решения на основе аппаратных средств в данной работе не рассматриваются.

Так как задача разрабатываемой системы – предотвращение столкновений с препятствиями, она не должна быть причиной столкновений, которых не произошло бы при её отсутствии. По этой причине траектория движения звеньев манипулятора не должна изменяться, чтобы не происходило столкновений с препятствиями, о которых неизвестно системе, но известно оператору. Кроме того, при отсутствии воздействия со стороны оператора манипулятор должен стоять неподвижно. Эти критерии исключают применение методов, основанных на потенциальном планировании.

Система также не должна ухудшать такие характеристики манипулятора, как точность и плавность работы. Так как алгоритмы, работающие в пространстве конфигураций, являются компромиссом между точностью и плавностью работы с одной стороны, и занимаемым объёмом памяти с другой, их применение для решения задачи также нецелесообразно.

Также учтём, что резкие остановки манипулятора могут привести к жёстким ударам, что способствует быстрому износу системы, поэтому система должна плавно уменьшать скорость движения манипулятора по мере приближения к препятствию. Это также позволит снизить требования к производительности системы, поскольку по мере уменьшения скорости движения звеньев манипулятора перемещения звеньев за одну итерацию будут уменьшаться, увеличивая точность системы, что позволяет изначально использовать более грубый шаг, то есть увеличить длительность одной итерации алгоритма.

В связи со сказанным выше, выберем следующую стратегию поведения: По мере приближения звена к препятствию его скорость замедляется таким

образом, чтобы время до вероятного столкновения было постоянным. При этом соотношения между составляющими скорости остаются неизменными, чтобы гарантировать неизменность траектории. Кроме того, вводится пороговое значение расстояния, и если обнаруживается, что на следующей итерации расстояние окажется меньше порогового, то манипулятор останавливается. Это необходимо для предотвращения соприкосновений звеньев манипулятора с препятствиями. Для реализации этой стратегии на каждой итерации необходимо вычислять расстояние между звеньями и препятствиями.

3.2. Выбор алгоритмов обнаружения и прогнозирования столкновений

Для реализации алгоритма предотвращения столкновений манипулятора с препятствиями выбрана двухэтажная иерархическая структура. На широкой фазе алгоритм выполняет обнаружение только уже произошедших пересечений упрощённых ограничивающих объёмов, а на узкой фазе прогнозирует столкновение детализированных ограничивающих объёмов, причём для обеспечения более точной аппроксимации к одному звену может быть привязано несколько ограничивающих объёмов.

Для обеспечения достаточного времени для торможения манипулятора после начала работы алгоритма узкой фазы ограничивающие объёмы широкой фазы включают в себя не только сами звенья, но и пустое пространство вокруг них (далее это пространство называется зоной безопасности). Размеры зоны безопасности выбирается исходя из максимально возможной линейной скорости движения среди всех точек звена. Пример аппроксимации звена с зоной безопасности представлен на рисунке 3.2.1.

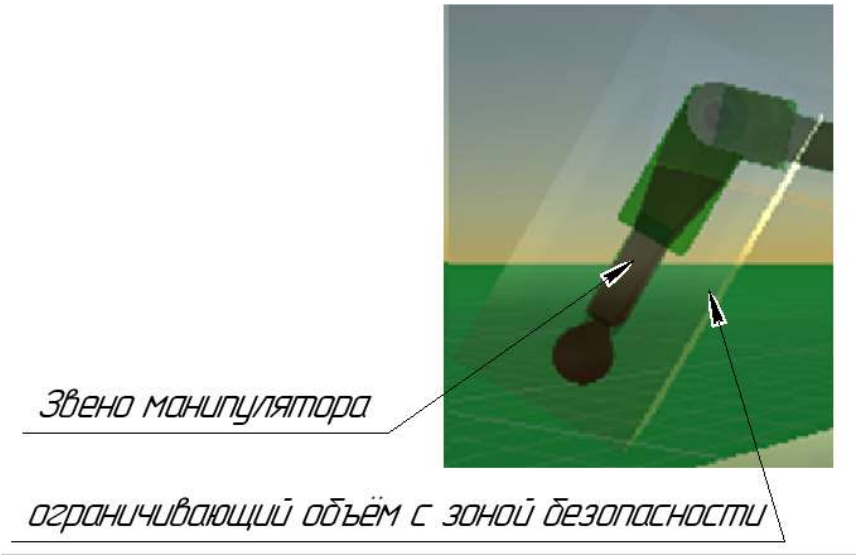

Рисунок 3.2.1 – Ограничивающий объём с зоной безопасности

На широкой фазе в роли ограничивающих объёмов выбраны OBB, так как из стандартных форм для ограничивающих объёмов они наиболее точно аппроксимируют продолговатые вращающиеся звенья манипулятора. Для обнаружения пересечений ограничивающих объёмов на широкой фазе используется алгоритм на основе SAT, так как этот алгоритм позволяет сделать это за константное время путём нахождения проекций на не более чем 15 направлений.

На узкой фазе для обеспечения замедления манипулятора вблизи препятствий необходимо получать информацию о расстоянии между объектами, поэтому будет использоваться алгоритм GJK. Поскольку геометрическая аппроксимация на этом шаге должна быть довольно точной, будет использовано три разных типа ограничивающих объёмов: сфера, цилиндр и OBB. При этом к одному звену можно будет привязать более одного ограничивающего объёма. Полная иерархия ограничивающих объёмов для одного звена представлена на рисунке 3.2.2.

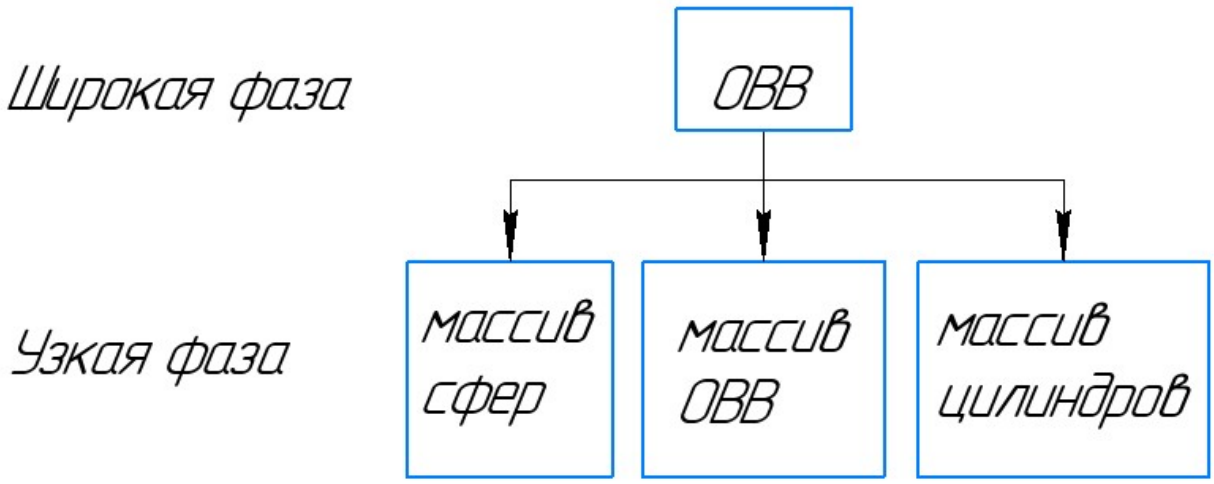

Рисунок 3.2.2 – Иерархия ограничивающих объёмов звена

Точное прогнозирование будущих столкновений является ресурсоёмкой задачей, поэтому целесообразно вместо этого вычислять приближённое время до столкновения. Поскольку угловые перемещения звеньев за одну итерацию алгоритма малы, они не учитываются и рассматриваются только линейные составляющие движения наиболее близко расположенных точек звеньев. Для пары выпуклых объектов, вектор расстояния между которыми известен, легко доказать, что необходимым условием столкновения при отсутствии вращения является равенство длины проекции вектора перемещения на направление вектора текущего расстояния величине, равной текущему расстоянию между ними. Доказательство представлено на рисунке 3.2.3.

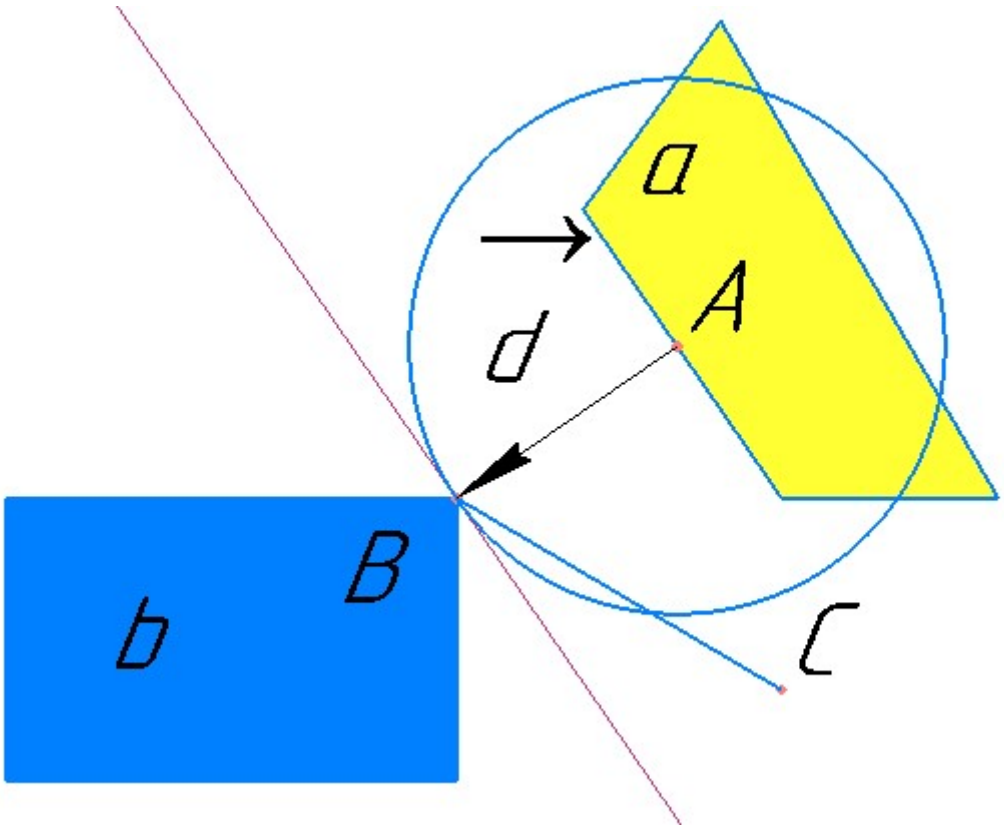

Рисунок 3.2.3 – Доказательство утверждения

Здесь $\vec{d}$ – известный вектор расстояния, A и B – ближайшие точки выпуклых объектов a и b. Построим окружность с центром в точке A радиуса d. Поскольку точка B – ближайшая к точке A точка, принадлежащая объекту b, внутри окружности нет точек, принадлежащих объекту b. Проведём касательную к окружности в точке B, которая одновременно является нормалью к вектору $\vec{d}$, поскольку он является радиусом окружности. Поскольку через лежащую на окружности точку проходит только одна касательная, любая другая прямая, проходящая через точку B, пересекает окружность. Тогда для любой точки C, лежащей с той же стороны от касательной, что и объект a, отрезок CB будет включать в себя хорду окружности, а так как внутри окружности нет точек, принадлежащих объекту b, а по определению выпуклого объекта отрезок между любыми двумя принадлежащими выпуклыми объекту точками не может выходить за пределы этого объекта, точка C не может принадлежать объекту b. Фактически это означает, что у объекта b не может быть ни одной точки, проекция которой на направление вектора $\vec{d}$ лежала бы

ближе к объекту a, чем точка B. Аналогично доказывается, что и у объекта a не может быть ни одной точки, проекция которой на направление вектора $\vec{d}$ лежала бы ближе к объекту b, чем точка A, то есть расстояние между проекциями объектов на направление вектора $\vec{d}$ равно d. Необходимым условием пересечения объектов является пересечение их проекций на любое направление, а потому для столкновения объектов проекция вектора их перемещения на направление вектора $\vec{d}$ должна быть не менее d, что и требовалось доказать.

С учётом вышесказанного принимается что минимально возможное время до столкновения – это время, затрачиваемое на относительное перемещение объектов на вектор, проекция которого на направление вектора кратчайшего расстояния равна расстоянию между объектами. Тогда расчётная схема для приближённого определения времени столкновения для плоского случая представлена на рисунке 3.2.4.

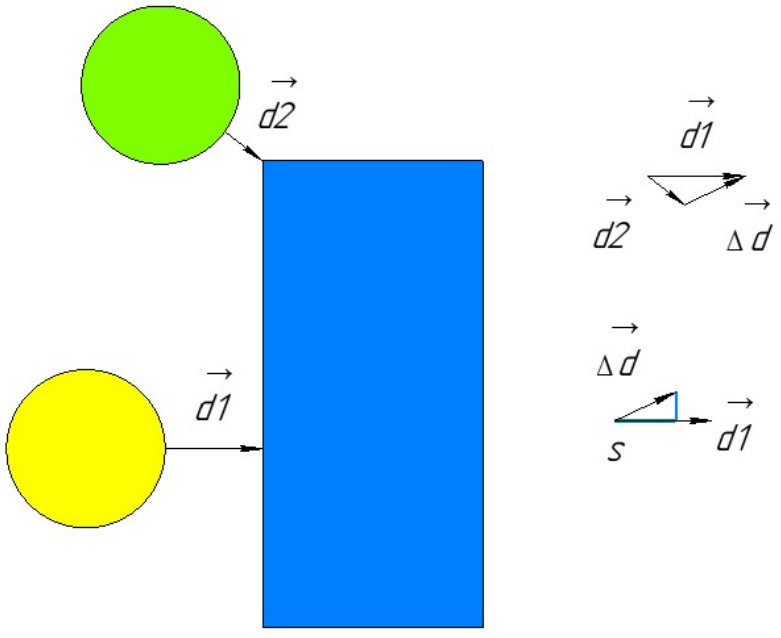

Рисунок 3.2.4 – Схема расчёта приближённого времени столкновения для плоского случая

Для определения времени используются векторы расстояния между ближайшими точками, полученные в результате работы GJK для двух положений манипулятора: текущего и положения на следующей итерации,

спрогнозированного из предположения, что скорость движения будет равна скорости, заданной оператором. На рисунке 3.2.4 текущему положению соответствует вектор $\overrightarrow{d_1}$, спрогнозированному следующему – вектор $\overrightarrow{d_2}$.

Вычислив вектор изменения кратчайшего вектора за одну итерацию $\overrightarrow{\Delta d}$ и спроецировав его на направление вектора текущего расстояния, можно получить величину, которая далее будет обозначаться сближением звеньев s (отрицательное значение свидетельствует о том, что звенья удаляются друг от друга, и тогда для этой пары объектов работа алгоритма завершена). Чтобы найти приближённое время до столкновения, разделим длину вектора текущего расстояния между объектами на сближение и умножим на время одной итерации.

Для обеспечения плавного снижения скорости по мере приближения к препятствию достаточно ограничить минимальное время до столкновения, а при обнаружении значения, меньшего чем установлено, умножать скорость на коэффициент замедления, равный отношению полученного времени до столкновения к допускаемому. Также необходимо учесть, что вектор расстояния будет равен нулю для всех точек, для которых обнаружено столкновение. В связи с этим при обнаружении столкновения в спрогнозированном положении точно определить сближение окажется невозможно, а потому для этого случая необходимо выполнить полную остановку манипулятора.

Такой подход также позволяет избежать проблемы туннелирования. Поскольку туннелирование подразумевает проход одного объекта сквозь другой за одну итерацию, фактически это означает, что сближение звеньев обязано быть больше расстояния между звеньями. В процессе выполнения программы это легко обнаружить, после чего манипулятор необходимо остановить, как и для случая обнаружения столкновения.

В соответствии с изложенным выше на выходе GJK будет приниматься не численное значение расстояния между объектами, а кратчайший вектор между

точками объектов. Поскольку данный подход справедлив только для выпуклых объектов, время до ближайшего столкновения должно рассчитываться для каждой пары ограничивающих объёмов в отдельности, после чего должно выбираться минимальное значение.

Также необходимо учитывать, что ограничивающие объёмы механически связанных звеньев будут находиться в состоянии пересечения в любой момент времени. В связи с этим применение описанных алгоритмов для обнаружения столкновений для этих пар звеньев не даст требуемого результата. Однако с учётом того, что в манипуляторах мобильных роботов обычно применяются кинематические пары пятого класса, для которых доступна только одна относительная степень подвижности, задача обнаружения и прогнозирования столкновений таких пар звеньев переходит в одномерное пространство. По этой причине для решения этой задачи достаточно указать предельные значения относительного угла поворота (или относительного перемещения) звеньев, и на основании данных о текущем значении угла и скорости его изменения вычислять время до столкновения звеньев.

Кроме того, целесообразно учитывать, что некоторые пары объектов не могут пересечься при любых конфигурациях манипулятора. По этой причине с целью сокращения затрачиваемого времени прогнозирование столкновений будет осуществляться только для пар объектов, указанных при инициализации.

4. Реализация алгоритма на практике и его тестирование

4.1. Выбор используемых инструментов разработки

Прежде всего необходимо осуществить выбор вычислительного устройства, которое будет использоваться для испытаний алгоритма. В данной работе будет использоваться микроконтроллер stm32f407. Основными причинами такого решения являются наличие у этого микроконтроллера аппаратной поддержки вычислений с плавающей запятой, широкое использование микроконтроллеров серии f400 в реальных роботах и наличие отладочной платы для проведения испытания.

Далее необходимо выбрать среду, которая будет использоваться для написания кода. Поскольку для различных семейств микроконтроллеров необходимо применение различных компиляторов, выбрана среда программирования keil, которая осуществляет поддержку микроконтроллеров stm32.

Следующей задачей является выбор языка программирования, который будет использоваться в ходе разработки. Поскольку вычисления, используемые в разрабатываемой системе, являются трудоёмкими, а производительность вычислительного устройства невысока, применять высокоуровневые языки программирования нецелесообразно. В то же время алгоритм является достаточно сложным, а потому используемый язык должен обеспечивать необходимые инструменты для удобных разработки и поддержки кода. Также в ходе работы происходит моделирование множества объектов, что делает удобным применение объектно-ориентированных языков программирования. Выбранная среда разработки предоставляет ограниченный выбор языков программирования. Исходя из всего вышеперечисленного, наиболее удобным языком является C++.

Для моделирования перемещения звеньев манипулятора в пространстве необходимо решить для манипулятора прямую задачу геометрического анализа. С целью обеспечения универсальности разрабатываемого алгоритма для решения задачи будут использоваться матрицы поворота и матрицы

перехода. Для этого целесообразно применить библиотеку линейной алгебры. Поскольку для обеспечения универсальности разрабатываемая система не должна быть привязана к конкретной архитектуре вычислительных устройств, применение библиотеки CMSIS-DSP, разработанной для устройств архитектуры ARM, нецелесообразно. Также немаловажными факторами являются широкая распространённость используемой библиотеки и наличие открытого исходного кода. Исходя из всего вышеперечисленного, была выбрана библиотека Eigen, предоставляющая необходимые классы – matrix, vector и transform – для которых реализованы основные алгебраические операции и операции преобразования пространства.

Поскольку для тестирования системы было решено обеспечить передачу данных на персональный компьютер (более подробно тестирование описано в главе 5), необходимо выбрать интерфейс передачи данных. Так как высокая скорость и высокая точность не требуются, целесообразно использовать интерфейс UART, так как он является наиболее простым. Для соединения отладочной платы с персональным компьютером использован преобразователь интерфейсов USB-UART CP2102.

Так же для симуляции пульта управления целесообразно подключить к отладочной плате кнопки, состояние которых определяет вектор введённой оператором желаемой скорости.

Для быстрой настройки периферии микроконтроллера и упрощения работы с ней применены библиотека hal и генератор кода cubeMX, поддерживающий выбранный микроконтроллер и выбранную среду разработки.

4.2. Описание структуры реализованной библиотеки

Для решения задачи в работе реализована библиотека, включающая в себя несколько классов и алгоритмов. Упрощённая UML – диаграмма библиотеки представлена на рисунке 4.2.1. На диаграмме не показана библиотека Eigen, так как она используется во всех классах и функциях. Также не показаны функции, используемые для инициализации объектов.

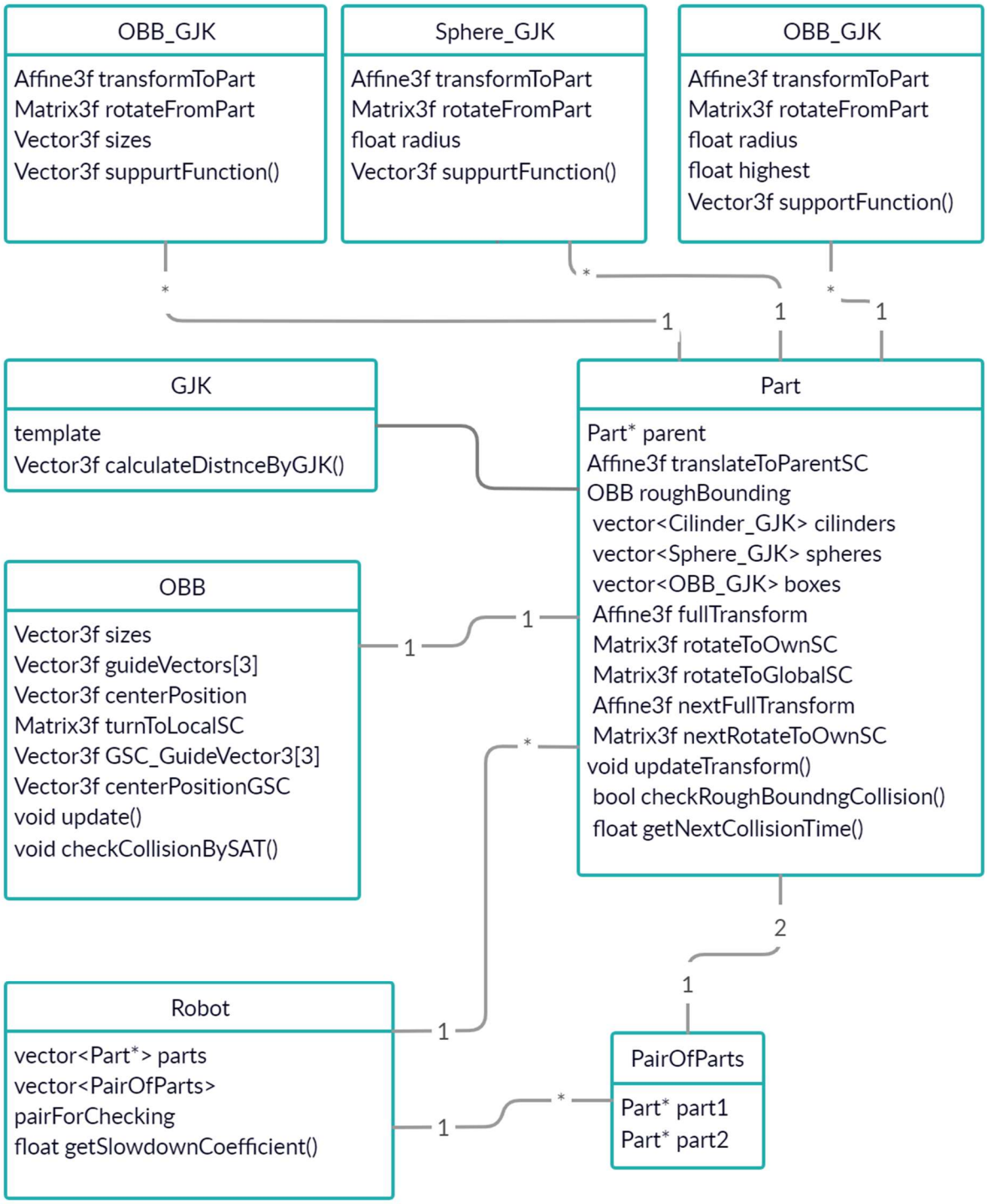

Рисунок 4.2.1 – Упрощённая UML – диаграмма библиотеки

Класс OBB используется на грубой фазе алгоритма. Объект этого класса содержит в себе три направляющих вектора, хранящих информацию о его ориентации в системе координат звена, вектор положения его центральной точки в системе координат звена, те же вектора в глобальной системе координат, а также вектор с размерами звеньев и матрицу поворота из

глобальной системы координат в систему координат ограничивающего объёма. Помимо функций, используемых при инициализации объекта класса, класс содержит функцию update, которая осуществляет обновление сведений о расположении объекта в глобальной системе координат, и функцию checkCollisionBySAT, которая проверяет факт наличия пересечения объекта с другим объектом этого же класса.

Классы OBB_GJK, Cylinder_GJK и Sphere_GJK используются в качестве аппроксимирующих объёмов в алгоритме GJK. Они содержат в себе матрицу поворота от системы координат звена к системе координат ограничивающего объёма, матрицу перехода от системы координат ограничивающего объёма к системе координат звена, а также информацию о размерах ограничивающего объёма. Помимо функций инициализации реализована функция supportFunction, то есть опорная функция для алгоритма GJK.

Класс Simplex, используемый в ходе работы алгоритма GJK, представляет из себя простой контейнер для не более чем четырёх объектов класса Vector3f. Он реализован с целью избежать применения контейнера std::vector, так как изменение его размера требует обращения к динамической памяти, а эта операция может занимать много времени.

Работа алгоритма GJK реализована без применения классов, поскольку такой подход был более удобен на стадии разработки. Модуль GJK содержит несколько основных функций. Функция calculateDistanceByGJK вызывается извне и возвращает вектор кратчайшего расстояния между двумя ограничивающими объёмами, для которых реализована функция suportFunction.

Функция evolveSimplex используется для итеративного изменения симплекса, в ходе которого он приближается к ближайшей к началу координат точке разности Минковского двух ограничивающих объёмов. Назначение функции getExtremalPoint – нахождение точки разности Минковского, наиболее удалённой в указанном направлении. Функция getNearestSimplexPoint используется для нахождения ближайшей к началу координат точки симплекса.

Класс Part описывает деталь вместе со всей иерархией аппроксимирующих объёмов. Функция updateTransform обеспечивает моделирование перемещения детали вдоль одной из её координатных осей или поворот вокруг одной из её координатных осей, причём в памяти хранится информация сразу о двух положениях: текущем и спрогнозированном следующем. Функция checkRoughBoundingCollision используется для проверки пересечения ограничивающих объёмов грубой фазы пары звеньев, а функция getNextCollisionTime возвращает приближённо вычисленное время до столкновения с другим звеном. Также объекты класса содержат информацию о максимально разрешённых значениях координаты. Этот класс также содержит в себе иерархию ограничивающих объёмов, то есть объект класса OBB и массивы с объектами классов OBB_GJK, Sphere_GJK и Cylinder_GJK.

Класс Robot используется для хранения взаимосвязей о порядке перестроения положения звеньев, а также обо всех парах звеньев, которые необходимо проверять на возможность столкновения (такой подход целесообразен, поскольку позволяет избежать проверки возможности столкновения для звеньев, которые не могут столкнуться ни в одной из возможных конфигураций манипулятора). Функция getSlowdownCoefficient этого класса используется для перестроения конфигурации манипулятора и для вычисления коэффициента замедления, необходимого для предотвращения ближайшего столкновения.

Полный текст исходного кода разработанной библиотеки доступен в открытом репозитории по адресу https://github.com/NikitaKhalyavin/NIR.git.

4.3. Схема тестирования алгоритма

Тестирование работы алгоритма осложнено необходимостью наличия оператора для введения вектора желаемой скорости, из-за чего время проведения тестирования ограничено. Однако замена оператора автоматизированными средствами тестирования проблематична из-за высокой сложности выполняемой задачи. Также важной задачей тестирования алгоритма является корректность его работы на микроконтроллере stm32f407.

На основании вышеизложенных факторов выбрана схема тестирования алгоритма. Ввод вектора желаемой скорости, а также оценка корректности работы алгоритма осуществляются оператором. Для этого ему необходимо устройство ввода, в роли которого применены кнопки, подключенные к отладочной плате. Состояние кнопок определяет вектор желаемой скорости. С целью контроля положения манипулятора оператору необходимо средство визуализации его расположения. Для этого в данной работе на базе среды разработки Unity была создана трёхмерная модель манипулятора. Сам алгоритм запускается на отладочной плате stm32discavery-f4, расположение и размеры ограничивающих объёмов для звеньев манипулятора задаются разработчиком вручную в соответствии с формой и размерами модели манипулятора. Также для удобства оператора ограничивающие объёмы изображены вокруг звеньев виртуального манипулятора в Unity. Фотография отладочной платы, преобразователя интерфейсов и кнопок управления представлена на рисунке 4.3.1, а модель манипулятора с ограничивающими объёмами представлена на рисунке 4.3.2.

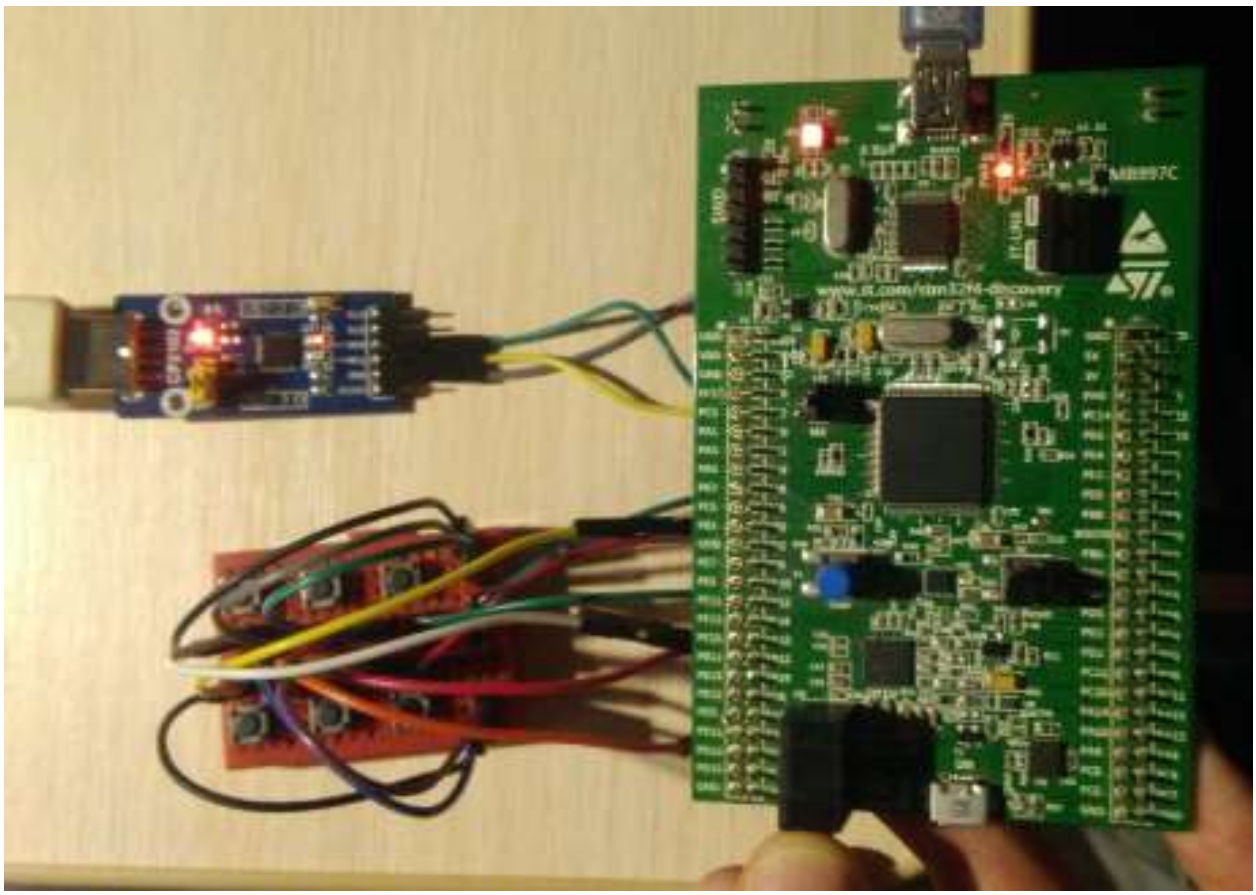

Рисунок 4.3.1 – Использованное оборудование

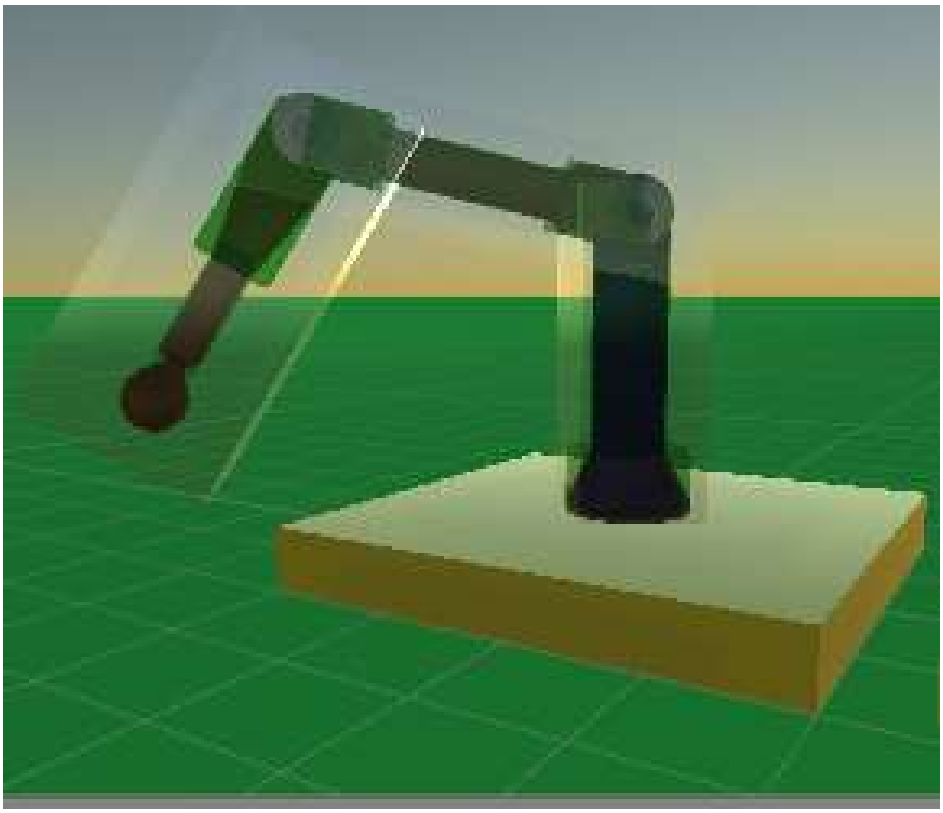

Рисунок 4.3.2 – Трёхмерная модель манипулятора в Unity

Для обеспечения связи между конфигурацией манипулятора в Unity и в отладочной плате плата передаёт через интерфейс UART обобщённые координаты звеньев. Unity изменяет конфигурацию виртуального манипулятора в соответствии с полученными данными. Кроме того, Unity имеет набор инструментов, подходящих для обнаружения пересечений трёхмерных объектов, которые могут быть использованы в качестве дополнительного средства обнаружения ошибок в работе алгоритма.

Поскольку Unity выполняет отрисовку кадров с ограниченной частотой, более частая отправка координат может привести к накоплению в буфере приёма множества необработанных значений и, как следствие, возникновению задержки между перемещением звеньев манипулятора и его визуализации в Unity. С целью предотвращения возникновения данной проблемы между итерациями алгоритма введена дополнительная задержка длительностью 40 мс.

Важной задачей тестирования является сбор сведений о времени, затрачиваемом на выполнение отдельных задач. Наиболее ресурсоёмкой является задача обнаружения столкновений, поэтому измерение времени будет осуществляться именно для неё. В ходе тестирования будем измерять максимальное, минимальное и среднее время работы алгоритмов SAT и GJK, а также определять суммарное затраченное на них время за продолжительный период работы алгоритма. Это позволит оценить пригодность разработанной библиотеки для решения поставленной задачи, а также на основе соотношений о затратах времени для этих двух алгоритмов определить наиболее перспективное направление для дальнейшей работы.

Для измерения времени работы функций в проект введён модуль time_measuring, который позволяет измерять время между событиями, а также собирать необходимые статистические сведения. При этом измерение времени возложено на аппаратный таймер микроконтроллера.

4.4.Результаты тестирования алгоритма

В ходе тестирования решались две основные задачи: оценка корректности работы алгоритма и оценка времени, затрачиваемого отдельными частями алгоритма, необходимая для определения направления дальнейшей оптимизации.

В ходе тестирования корректности работы оператор производил управление виртуальным манипулятором, периодически направляя манипулятор в сторону препятствия. В данных условиях разработанная программа успешно справлялась с задачей предотвращения столкновений с препятствиями, однако была выявлена другая проблема: при удалении от препятствия скорость манипулятора возрастает скачком до максимального значения, что может привести к чрезмерной нагрузке на приводы манипулятора.

Результаты измерения статистических значений времени, затрачиваемого на однократный вызов функций, при компиляции программы без оптимизации для некоторого продолжительного времени представлены в таблице 4.4.1.

Таблица 4.4.1 – Статистические сведения о временных затратах для основных алгоритмов при компиляции без оптимизации

Алгоритм	SAT	GJK
Максимальное время, мкс	1538	15400
Минимальное время, мкс	67	2010
Среднее время период измерений, мкс	356	8900
Количество вызовов за период измерений	4554	7684
Общее время за период измерений, мкс	$1{,}62 \cdot 10^6$	$6{,}83 \cdot 10^7$

Как видно из таблицы, среднее время работы GJK на два порядка больше среднего времени работы SAT. Хотя алгоритм GJK за период измерений вызывается реже чем SAT, общее время, затрачиваемое алгоритмом GJK

примерно в 17 раз больше. При этом судя по данным о среднем времени, затрачиваемом алгоритмами, и частоте их вызова в пределах одной итерации, за одну итерацию алгоритма может быть затрачено более 40 мс, что приведёт к заметным глазу колебаниям скорости манипулятора и скажется на комфорте работы оператора.

Результаты измерения статистических значений времени, затрачиваемого на однократный вызов функций, при компиляции программы с использованием оптимизации среды keil третьего уровня для некоторого продолжительного времени представлены в таблице 4.4.2.

Таблица 4.4.2 – Статистические сведения о временных затратах для основных алгоритмов при компиляции с оптимизацией

Алгоритм	SAT	GJK
Максимальное время, мкс	266	3290
Минимальное время, мкс	12	328
Среднее время период измерений, мкс	70,4	1540
Количество вызовов за период измерений	$2 \cdot 10^4$	$3{,}79 \cdot 10^4$
Общее время за период измерений, мкс	$1{,}41 \cdot 10^6$	$5{,}85 \cdot 10^7$

Как видно из таблицы, при использовании оптимизации время работы обоих алгоритмов существенно сократилось. Однако среднее время работы алгоритмов и частота их вызова по-прежнему не позволяют с уверенностью утверждать, что работа алгоритма не вызовет заметных глазу колебаний скорости манипулятора. Тем не менее, в ходе тестирования заметные колебания скорости виртуального манипулятора зафиксированы не были. Среднее время работы GJK в 22 раза больше среднего времени работы SAT. Общее же время, затрачиваемое алгоритмом GJK примерно в 40 раз больше.

Таким образом, из результатов тестирования очевидно, что оптимизация алгоритма SAT или введение дополнительной ступени иерархии

ограничивающих объёмов выше SAT не приведут к существенному ускорению работы системы. Дальнейшее ускорение алгоритма должно осуществляться посредством ускорения работы алгоритма GJK или снижением частоты его вызова.

5. Заключение

В ходе аналитического обзора существующих способов решения проблемы было рассмотрено три разных подхода, ни один из которых не удовлетворял поставленным требованиям. В результате была выработана новая стратегия поведения, подходящая под все выдвинутые требования и разработана программная реализация для этого подхода. В результате тестирования этой реализации на микроконтроллере stm32f407 было выявлено, что реализация в целом справляется с поставленной задачей, однако нуждается в дальнейшей доработке с целью предотвращения скачков скорости.

Сравнение времени, затрачиваемого на выполнение отдельных частей алгоритма, позволило определить наиболее приоритетные направления дальнейшей работы в области оптимизации алгоритма:

- уменьшение длительности работы алгоритма GJK (или использование другого способа вычисления кратчайшего вектора, например аппроксимации объектов воксельной карты поверхности [14]);
- снижение частоты обращения к алгоритму путём изменения иерархической структуры ограничивающих объёмов.

Также разработанное решение требует более детального тестирования с использованием манипуляторов различных кинематических схем и различного количества препятствий. В ходе тестирования также необходимо произвести более детальное моделирование поведения манипулятора с учётом перемещения звеньев манипулятора за время вычислений и динамики движения звеньев.

Наконец, разработанную систему необходимо протестировать на реальном манипуляторе с целью удостовериться в правильности применённой модели тестирования.

6. Список литературы

1. Pre-Contact Sensor Based Collision Avoidance Manipulator – Ahmad Zaki Hj Shukor, Ng Jack Kii, Muhammad Fahmi Miskon, Fariz Ali@Ibrahim, Muhammad Herman Jamaluddin, 2017
2. Sensor covering of a robot arm for collision avoidance - D. Gandhi, Enric Cervera, 2003
3. A 3-dimensional force field method for robot collision avoidance in complex environments – P. Chotiprayanakul, D. K. Liu, D. Wang, G. Dissanayake, 2007
4. An improved artificial potential field method of trajectory planning and obstacle avoidance for redundant manipulators – Wenrui Wang, Mingchao Zhu, Xiaoming Wang, Shuai He, Junpei He, Zhenbang Xu, 2018
5. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots – Oussama Khatib, 1986
6. On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: an industrial use case – Mohammad Safeea, Pedro Neto, Richard Bearee, 2019
7. Efficient Computation of Configuration Space Transforms for Collision free motion planning - Rachit Sapra, Somajyoti Majumder, Michael Mathew, 2014
8. Определение коллизий аппроксимирующих сфер и прямоугольных параллелепипедов в системах трехмерного моделирования - Трушин А. М., 2015
9. 3D Collision Detection: A Survey Pablo Jimenez, Carme Torras, Federico Thomas, 2001
10. Анализ столкновений – Горячкина А. Ю., 2014
11. Алгоритмы обнаружения столкновений – Д. И. Собинов, В.В. Коробицын, 2010
12. Collision Detection Algorithms for Motion Planning – P. Jim´enez, F. Thomas and C. Torras, 1998

13. A New Fast and Robust Collision Detection and Force Computation Algorithm Applied to the Physics Engine Bullet: Method, Integration, and Evaluation – Mikel Sagardia, Theodoros Stouraitis, and João Lopes e Silva, 2014
14. New trends in collision detection performance – Quentin Avril, Valérie Gouranton, Bruno Arnaldi, 2009
15. The Gilbert-Johnson-Keerthi Distance Algorithm – Patrick Lindemann, 2009
16. Technical Report TR-TI-2005-1-Collision Detection for k-DOPs using SAT with Error Bounded Fixed-Point Arithmetic - Stefan Hochgürtel, Andreas Raabe, Joachim K Anlauf, Gabriel Zachmann, 2005